Smart IoT Based
Water and Fertilization System

D. S. R. Shehan

2024

Smart IoT Based Water and Fertilization System

A dissertation submitted for the Degree of Master of Information Technology

D. S. R. Shehan
University of Colombo School of Computing
2024

Declaration

Name of the student: D S R Shehan
Registration number: 2019/MIT/090
Name of the Degree Program: Master of Information Technology
Project/Thesis title: Smart IoT Based Water and Fertilization System

1. The project/thesis is my original work and has not been submitted previously for a degree at this or any other University/Institute. To the best of my knowledge, it does not contain any material published or written by another person, except as acknowledged in the text.

2. I understand what plagiarism is, the various types of plagiarism, how to avoid it, what my resources are, who can help me if I am unsure about a research or plagiarism issue, as well as what the consequences are at University of Colombo School of Computing (UCSC) for plagiarism.

3. I understand that ignorance is not an excuse for plagiarism and that I am responsible for clarifying, asking questions and utilizing all available resources in order to educate myself and prevent myself from plagiarizing.

4. I am also aware of the dangers of using online plagiarism checkers and sites that offer essays for sale. I understand that if I use these resources, I am solely responsible for the consequences of my actions.

5. I assure that any work I submit with my name on it will reflect my own ideas and effort. I will properly cite all material that is not my own.

6. I understand that there is no acceptable excuse for committing plagiarism and that doing so is a violation of the Student Code of Conduct.

Signature of the Student	Date
	06-10-2024

Certified by Supervisor(s)

This is to certify that this project/thesis is based on the work of the above-mentioned student under my/~~our~~ supervision. The thesis has been prepared according to the format stipulated and is of an acceptable standard.

	Supervisor 1	**Supervisor 2**	**Supervisor 3**
Name	Malik Silva		
Signature			
Date	14th Oct 2024		

ABSTRACT

This project describes the design, development and evaluation of a Smart IoT Based Water and Fertilization System that utilizing IoT technology. Its goal is to address issues, in agriculture such as reducing water usage, improving fertilizer efficiency and promoting precision agricultural methods. The system integrated with sensors to monitor soil moisture, temperature, humidity and nutrient levels using IoT technology. This enables adjustments of watering and fertilization based on the requirements of each crop.

The innovative Smart IoT Water and Fertilizer System is designed with a technological foundation that encompasses both front-end and back-end components to ensure seamless operation and user interaction. Angular is utilized for creating a dynamic and responsive user interface while Redux efficiently manages the applications state on the end. On the backend the system uses .NET Core for its services, Entity Framework for object mapping, PostgreSQL as database for data handling SignalR for real-time communication and MQTT, for IoT messaging. This system is hosted on Azure offering data management capabilities, real time communication features and user-friendly interface.

The system has evaluated with a survey and a small-scale pilot project. Based on the results, a few modifications were made to algorithms to enhance the accuracy and efficiency of the system. The project will be extended to the next stage of development to include a mobile app and a rainwater utilization feature with the capability to handle more crops.

ACKNOWLEDGMENTS

My deepest gratitude is for the School of Computing of the University of Colombo. In addition to providing me with the necessary academic resources, the university has been an invaluable resource for personal and professional development. The lecturers and staff provided invaluable support, offering direction and insights crucial to this project and academic growth.

I would like to express my sincere appreciation to MR. Malik Silva, my project supervisor, for providing invaluable support during this project. Without his assistance, the successful completion of this project would have been far more challenging.

Finally, I express my gratitude to my family members, friends from Cloud Solution International (Pvt) Ltd, and all those who contributed to my success in the project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

GDP - Gross domestic product

IoT – Internet of Things

UI – User Interface

UX – User Experience

API - Application Programming Interface

MQTT - Message Queuing Telemetry Transport

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

CRUD - Create, read, update and delete

UML - Unified Modeling Language

ORM - Object Relational Mapper

ER - Entity Relationship

HTML - Hypertext Markup Language

SCSS - Syntactically Awesome Style Sheets

USB - Universal Serial Bus

RH - Relative Humidity

NPK - Nitrogen, Potassium, Phosphorous

RAM – Random Access Memory

1 CHAPTER – INTRODUCTION

1.1. Problem Overview and Motivation

The agriculture sector in Sri Lanka significantly contributes to both the economics and culture of the country. Over 30% of the population is involved in agriculture activities. Agriculture contributes to about 7.5 percent of the nation's GDP, and it has a significant impact on the livelihoods of many citizens in the country. Sri Lanka's agricultural sector continues to rely heavily on traditional farming methods. Typical agricultural practices frequently result in resource waste and environmental degradation. Practices such as flood irrigation and manual labor can result in an excessive use of water and fertilizers, compromising soil health and water availability.

Sri Lankan farmers are facing following challenges in recent years.

Water scarcity: Sri Lanka faces challenges with water scarcity due to a combination of factors such as increasing population, climate change, and deforestation. This has resulted in reduced water resources availability for agriculture, making it difficult for farmers to access the water they need for their crops (Chandrasekara, 2021). As per the world health program, 80% of Sri Lanka's land is in dry and intermediate zones which face frequent water shortages. Sri Lanka's dry zone is the main paddy producing area in the country and some parts of this area will face an absolute scarcity of water by 2025. Furthermore, research has suggested that paddy production in Sri Lanka will increase by 10% by year 2025 and that additional amount will be totally irrigation based (Rodrigo, 2013).

Soil Degradation: Overuse of fertilizers can lead to soil degradation, reducing the soil's fertility and loss of biodiversity. When too many fertilizers are used, the extra nutrients can leach into the groundwater and bodies of water on the surface. This can negatively impact aquatic environments and lead to the growth of harmful algae. Excess nutrients in the soil can change its pH levels that can reduce plant growth (Bisht & Chauhan, 2020).

Crop Yields: Overusing water and fertilizers could result in reduced crop production impacting the income of farmers and food security, in the country. Presently 15% of the people, in Sri Lanka are identified as experiencing food insufficiency (Thibbotuwawa, 2021).

Lack of access to modern technologies: Many farmers in Sri Lanka face challenges in accessing farming technologies like precise irrigation systems and real time soil monitoring systems. Utilizing these tools is essential, for enhancing their techniques to increased crop production and the adoption of eco-friendly approaches. (Thibbotuwawa, 2021).

Environmental Impact: Overusing water and fertilizers can have effects on the environment by polluting water sources.

The aim of this project is to develop and implement a solution, for managing water and fertilizer that combines Internet of Things (IoT) to tackle the issues mentioned earlier. The system will continuously track soil moisture, nutrition levels, temperature and humidity levels and provide automation on watering and fertilizing in real-time. Machine learning algorithms will analyze sensor data to identify trends and predict the timing and amount of watering and fertilization needed.

1.2. Objectives

The following objectives will be considered when planning and implementing the system to use water and fertilizers more efficiently,

- Develop a smart water and fertilization system that uses sensors and IoT technology to monitor soil moisture, nutrient levels and other environmental conditions, then use this data to optimize water and fertilizer usage in crop production.
- Integrate the smart system with a user-friendly web platform that allows farmers to oversee and manage watering and fertilization from a distance, in real time. They can also get notifications and suggestions tailored to crop conditions and weather forecasts.
- Integration with machine learning algorithms to enable the system to make more accurate predictions and recommendations based on crop and environmental data.

Machine learning integration will be a huge advantage to system capabilities to make predictions with the historical data. These predictions beneficial for farmers to make decisions on the plant as well as environment variables.

1.3. Background of the Study

The Internet of Things (IoT) is revolutionizing agriculture worldwide giving rise to concepts, like "Smart Farming" and "Precision Agriculture." Technology is playing a role in the industry by providing farmers with data driven insights to improve efficiency, productivity and sustainability. However, the adoption of technologies in Sri Lanka has been lacking recently. Many farmers in Sri Lanka have yet to embrace cutting edge farming technologies such, as precision irrigation systems and real time soil sensors. These innovations have the potential to increase crop yields minimize impact and optimize water and fertilizer usage effectively. The primary focus of this initiative is to introduce and integrate IoT based systems into Sri Lankas sector.

1.4. Scope of the Study

- The proposed project is a smart water and fertilization system that connect both hardware and software components. Microcontrollers with sensor technology will be used to capture parameters such as NPK levels, soil moisture, temperature, and humidity.

- The system will have a web application that farmers can access data from the sensors, and automate irrigation and fertilization processes. Based on collected parameters, the system will determine how much water and fertilization should be applied to crops. Additionally, the system capable in both manual and automated watering and fertilization process.

- The data gathered by the sensors will be combined with machine learning to generate more precise forecasts and suggestions. The machine learning process will be based on supervised learning, and a model will be trained using a collection of known data to make accurate predictions.

- At this phase, the project will concentrate on a single crop, for which all hardware components will be constructed. But the web application will support numerous crop types.

- To ease the implementation of this project, the irrigation and fertilization control flow will be indicated using relays and pumps instead of actual water and fertilizer pumping. procedures, which require more time and resources to perform.

1.5. Structure of the Dissertation

The rest of this dissertation has following structure:

- Chapter 2: Background

  This chapter will include a requirement analysis of the project. Further, it will review and analyze similar systems and their key points.
- Chapter 3: Design

  The design chapter outlines the technical design of the system architecture, hardware setup, and choice. of sensors to algorithms, UI/UX design, and diagrams.
- Chapter 4: Implementation

  This chapter describes implementation, configurations, software, and hardware integrations. best practices, and major code segments of this project.
- Chapter 5: Testing and Evaluation

  This chapter will provide an evaluation of the project's accuracy, effectiveness, and quality assurance of the final product.
- Chapter 6: Conclusion

  This chapter describes the major key areas of the project and further discusses the overall result of the system.

2 CHAPTER – BACKGROUD

2.1. Requirement Analysis

2.1.1. Hardware Requirements

Designing a successful IoT-based system requires a thorough consideration of the hardware requirements. These are the tangible, physical components that form the system's backbone, including sensors that capture vital environmental data, microcontrollers that analyze this data, and actuators that respond based on the interpreted data. This technology interacts in a continuous data collection, processing, and action cycle, enabling intelligent irrigation and fertilization to optimize crop health and yield, following the IoT component required to design the hardware setup.

Micro Controller

This is the main component of the hardware setup. The microcontroller is responsible for fetching sensor readings, processing collected data and transmitting information to the application. In return, it executes instructions given by the application. There are various types of microcontrollers which can be used for IoT solutions, such as Raspberry Pi, Arduino, ESP.

ESP32 wroom-32 will be used for the system because of the following reasons.

The ESP32 wroom-32 board is one of the most well-known in the ESP series. It is excellent for beginners and small to medium-sized projects. It has a powerful set of tools that works reliably. Since it has enough digital and analog input/output pins, it can connect all the necessary sensors and actuators for this setup. The ESP32 development board comes with built-in Wi-Fi support. Therefore, additional integration is not required for internet usage.

Soil Moisture Sensors

The main function of this sensor is to measure water content in the soil. There are two types of soil moisture sensors.

- Resistive Soil Moisture Sensor - This sensor operates on the principle that water is an excellent electrical conductor. Two probes are inserted into the soil by the sensor. The resulting current is measured when a voltage is applied to the sensors.
- Capacitive Soil Moisture Sensor - Instead of measuring resistance, these sensors measure the soil's dielectric permittivity, which varies depending on its moisture content.

The data obtained from the soil moisture sensor can be applied to determine the ideal time and quantity of watering for the plants. The system efficiently controls moisture levels, preventing excessive water consumption and promoting better growth of plants.

Temperature and Humidity Sensor

The temperature and humidity sensor measures the temperature and humidity levels in the environment where the system is installed. Typically, the sensor will have a digital output that can be easily interfaced with a microcontroller. Some sensors may necessitate the use of an analog-to-digital converter. These sensors' data can be utilized to make irrigation and fertilization decisions. For instance, higher temperatures may require more frequent watering, whereas high humidity may require less irrigation. Similarly, the application rate of fertilizer may be modified based on these parameters.

Actuators

These components perform physical tasks based on data received from the microcontroller. Actuators include relays, water pumps, and fertilizer pumps.

Connectivity

The IoT device requires an internet connection. This could be accomplished via Wi-Fi, cellular networks. This enables the transmission of sensor data to the application and receiving instructions.

2.1.2. Software Requirements

The software includes an interface that allows farmers to engage with the system, monitor real-time information, and obtain recommendations for watering and fertilization. The software requires a database for storing the gathered sensor data, machine learning algorithms for analyzing the data, and application programming interfaces (APIs) for facilitating interaction with the hardware components.

Communication Protocols: Protocols like MQTT, HTTP/HTTPS will be needed for data transmission between devices and the UI and Backend.

Embedded Software: This includes the IoT device's firmware, which takes sensor data, processes it, controls the actuators, and sends the data to the communication protocol. Firmware should consist of MQTT consumers and publishers.

Backend Service: Required server-side service to manage underlying infrastructure, requests from client applications, and providing appropriate responses and communication with IoT.

Data Processing and Analysis: Required Framework to process and analyze using machine learning algorithms to make predictions.

User Interface - Web or mobile application, where users can view and interact with the system.

2.1.3. User Requirements

The primary user of the system is the farmer. Therefore, the user required a reliable, easy-to-use solution to optimize watering and fertilization practices. The system should provide real-time data and watering and fertilization automation. The system should be simple to install, maintain, and use, without requiring extensive technical expertise.

2.1.4. Functional Requirements

- **Continuous Monitoring**: The system should monitor soil conditions continuously using the installed sensors.
- **Automated Control**: The system should be able to automate the watering and fertilization process based on sensor values and system recommendation.
- **Real-time Data**: The system should operate real-time for water and fertilizer applications based on the sensor data and all information should appear real-time in user interface.
- **Data Processing and Analysis**: System should capable to process analyze sensor values and generate predictions with the help of machine learning algorithms.

2.1.5. Non-Functional Requirements

- **Reliability:** The system should provide accurate data and recommendations with minimal errors.
- **Efficiency:** The system should process the sensor data and perform tasks quickly.
- **Usability**: The user interface should be attractive and easy to navigate.
- **Scalability**: The system should be able to handle more plants without performance deration.

2.2. Review of Similar Systems

The MyJohnDeere Operations Center (John Deere, 2024), is an online tool that offers farmers immediate updates on their farming activities. It includes data on soil moisture, crop yield and machinery efficiency. Farmers can oversee and control their machinery, track their crops progress and make well informed choices about planting, fertilizing and harvesting. The platform comes equipped with various valuable functions.:

- Data collection: The platform integrates with numerous sensors and machinery to collect data on soil moisture, yield, and other vital metrics.
- Data analysis: Collected data will be further analyze and provide predictions based on machine learning algorithms.
- Equipment management: Farmers can use the tool to track their machines' performance and when they need to be serviced.
- Crop management: The platform has tools for crop planning, like picking seeds and figuring out how much fertilizer to use.
- Collaboration: Farmers can share statistics with their advisors and partners through the platform, which helps them work together and make better decisions.

The main benefits of the MyJohnDeere Operations Center are:

- **Enhanced decision making:** The system offers farmers, up to date insights, into their farming activities empowering them to decide on planting, fertilizing and harvesting practices.
- **Increased efficiency**: Farmers have the opportunity to enhance their productivity by adjusting their planting and fertilization practices according to, up to date information. This approach can lead to yields, cost savings and reduced waste, on the farm.

  Few user interfaces of the product are showing in figure 2.1 and 2.2

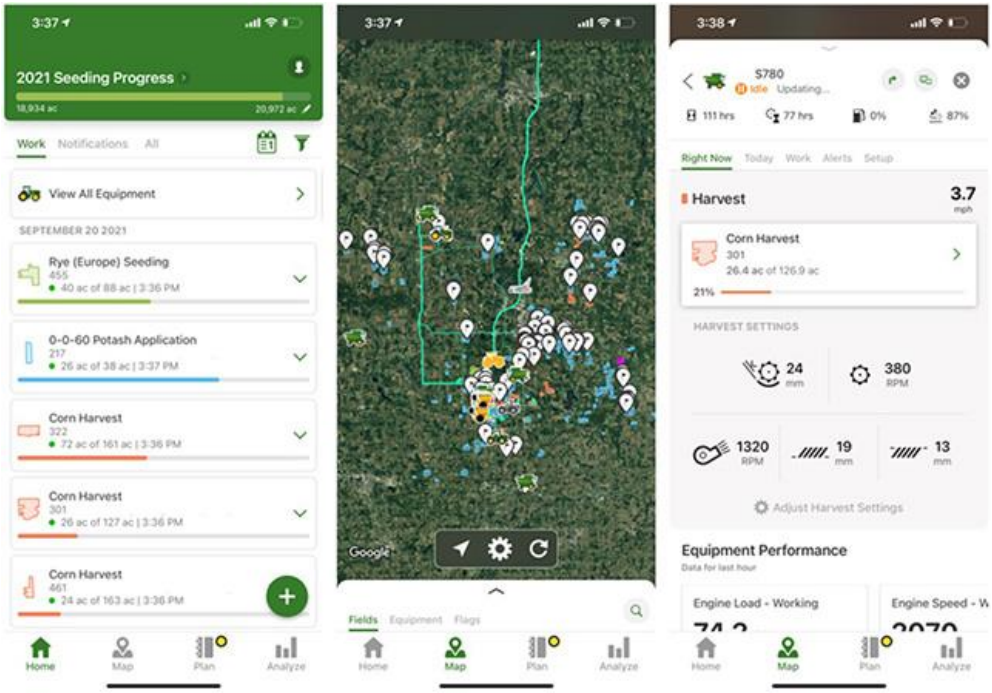


Figure 2.1: Mobile User Interface of MyJohnDeere

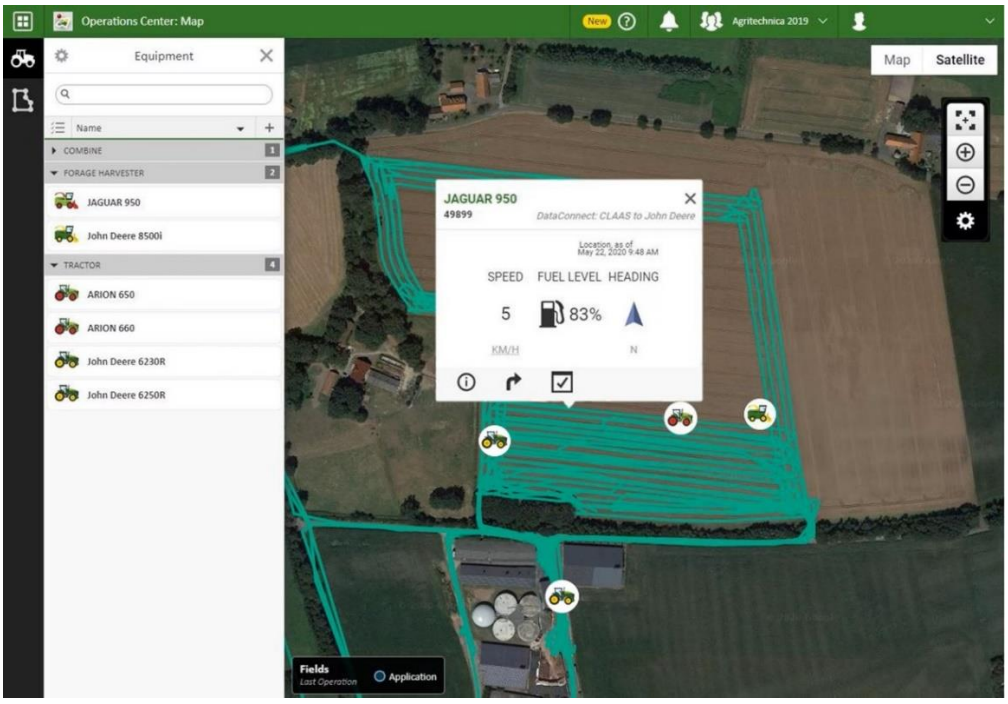


Figure 2.2 : Dashboard of MyJohnDeere

GroGuru

GroGuru (GroGuru, 2024) is an irrigation and soil monitoring technology that provide recommendations to farmers in optimizing water usage and increasing crop yield. The system collects real-time data on soil moisture, temperature using wireless soil sensors. This information will be sent to a cloud-based to provide insights to users.

GroGuru's "Root Zone Monitoring" is a unique feature that collects vital information about the condition of the root zone of crops. The root zone is the area of soil that encompasses the roots of a plant. It is important to plant health because this is where plants absorb most of their water and nutrients.

But GroGuru's system doesn't provide automation for applying fertilizer. User required to perform scheduled tasks manually.

Growlink Smart Irrigation Controller

The Growlink Smart Irrigation Controller (Growlink, 2024) can control both watering and fertilizing processes. It uses multiple sensors that send data in real-time, such as readings for soil moisture, temperature, and electrical conductivity (EC). These data will be used to make proper watering and fertilization process.

Growlink's system provide a mobile app to their users that can handle the system from anywhere. Based on sensor data, it has automated tools that change the watering and fertilizing plans.

Growlink Smart Irrigation Controller have some advantages and disadvantages as follows:

Advantages

- Provides comprehensive irrigation and fertilization management.
- Utilizes multiple sensors (soil moisture, temperature, and EC) for real-time data-driven precise control.
- Includes a mobile application for remote control and a simple data visualization interface.
- Provides automation features that modify irrigation and fertilization schedules in response to sensor data.

Disadvantages

- The system may require more work to configure and manage, particularly for non-technical users.
- Due to the advanced features and multiple sensor connections, the price could be higher.

Few user interfaces of the Growlink product are showing in figure 2.3 and 2.4

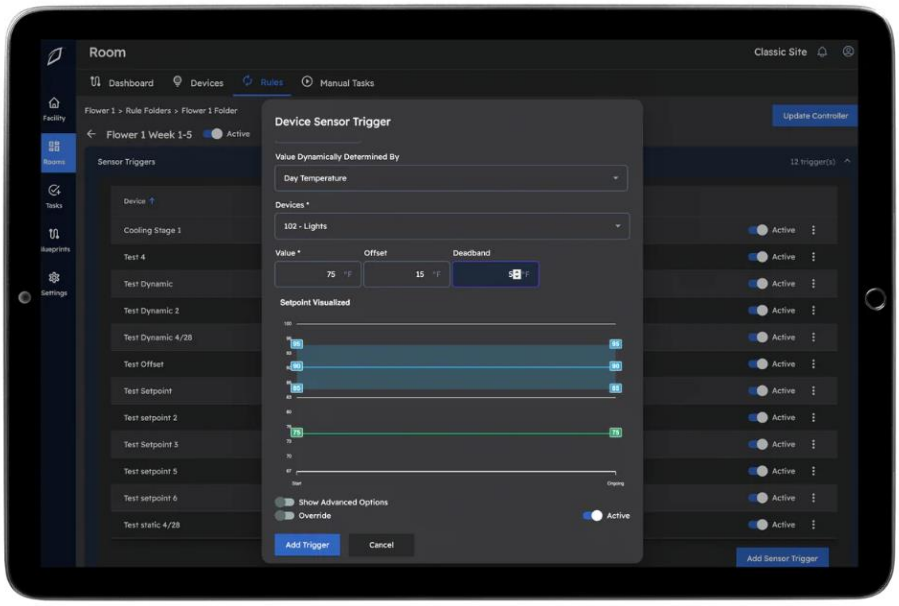


Figure 2.3 Dashboard of Growlink smart irrigation system

Figure 2.4: Desktop and Mobile App of Growlink System

2.3. Related Technologies

The system uses several cutting-edge technologies to make the user experience seamless, reliable, and effective.

Hardware Components: ESP-wroom-32 will be used as a microcontroller with sensor technology. 12V Water and fertilizer pumps will be used to indicate flows.

MQTT Broker: MQTT (Message Queuing Telemetry Transport) is a lightweight publish and subscribe messaging protocol that was designed to be used in low-bandwidth and unreliable network environments. MQTT Broker will be used as communication protocol between Microcontroller and server. All data transmission will be handled real time by MQTT broker.

Web Application Architecture: Proposed web application will be developed to Client Server Architecture. Server consist of the database and client-side users can use the system with web browser without using any software additionally on their local machines. Server and client sides will communicate and share data through Web API.

Technology Stack: Proposed system consists of web-based application, and it will be developed using Angular frontend, NET Core backend, Bootstrap, Web API 2.0 and Entity Framework, Postgres as Database.

Machine Learning – ML.NET will be used as machine learning framework.

Deployment: Web Application will be hosted on Azure services.

Layered architecture will be used as back-end architecture with asynchronous programming. Web API will be used for presentation tier; logic tier and data tiers will be implemented using a repository pattern. Entity framework will be used with Postgres server to reduce development time and make CRUD operations efficient.

2.4. Development Methodology

In software engineering, a software development methodology or system development methodology is a framework used to organize, plan, and manage the process of constructing an information system. Among other techniques, the classic Waterfall Model would be utilized due to the following key benefits.

- The project has a predefined timeline, and both development and implementation must be completed within that timeframe.
- All project milestones are established prior to the initiation of the project. Therefore, it is important to capture project requirements clearly and ensure they remain unchanged until the project is delivered.

3 CHAPTER – DESIGN

3.1. Introduction

This chapter provides an overall detailed description of the system's architecture. System architecture design is the process of organizing and structuring the entire system to meet the requirements, and future scalability needs while keeping the system maintainable and adaptable. It encompasses design techniques and methodologies applied to structures and solutions.

This system's architectural design provides a structural layout that specifies the system's components and how they interact with each other. Components are the system's fundamental pieces, including software modules, databases, user interfaces, etc.

This process contains graphical representations, diagrams, and notations such as UML to visualize and document the system's structure and processes.

3.2. System Architecture

The system is divided into two main parts: hardware setup and web application. In the hardware setup, all controllers and sensors will be controlled via a microcontroller, and the microcontroller communicates to the web application via the MQTT broker. Web applications consist of client servicer architecture and will communicate via HTTP API calls. High-level system architecture is shown in Figure 2.5

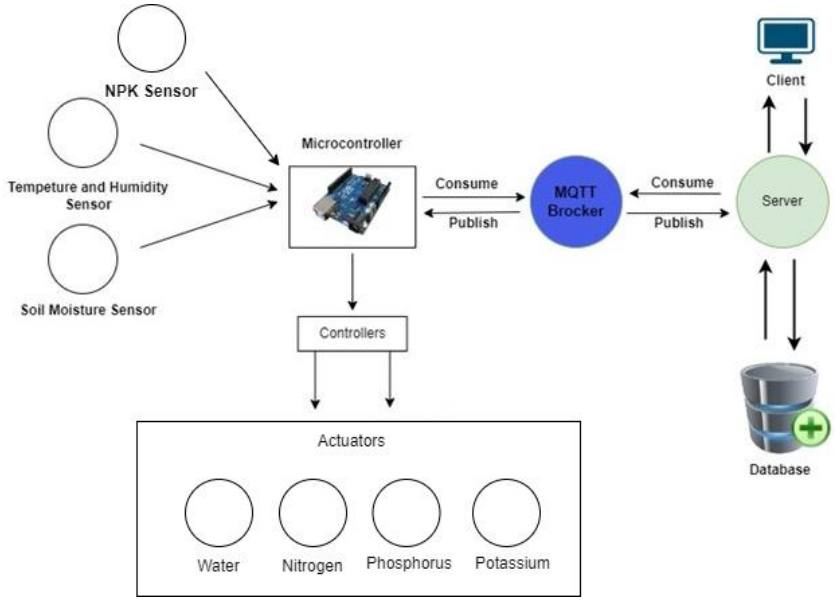


Figure 3.1 High-Level System Architecture

3.3. Frontend Architecture

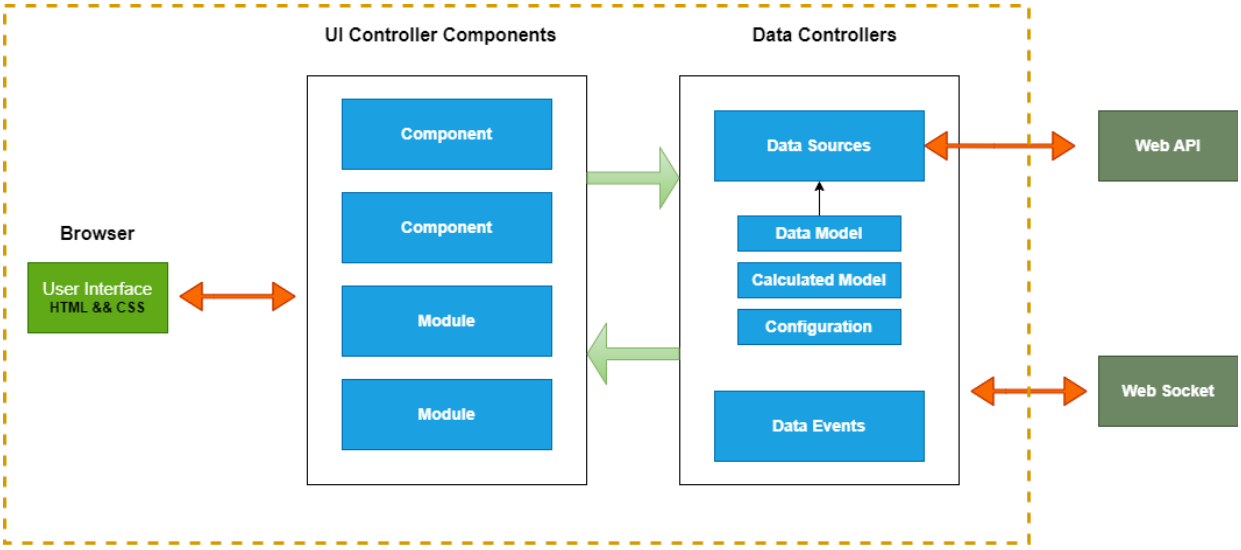


Figure 3.2 Frontend Architecture

Figure 3.2 shows the High-Level architecture of the front-end. The front-end application will be implemented on top of the Angular JavaScript framework. Angular is a prominent framework for developing web applications. It encourages using patterns and best practices for building scalable and maintainable applications. We refer to the organizational strategies and patterns used for Angular applications when considering Angular front-end architectures. Here are several common architectures and strategies which will be used in implementation:

Component-Based Architecture: Angular relies mainly on component-based architecture. This implies that UIs are made up of multiple components, each responsible for a unique feature. Components encapsulate the layout, data, and behavior of an interface element.

Modules: Angular groups related components, directives, pipelines, and services using modules (NgModules). Modules facilitate application organization and promote features such as lazy loading.

Services and Dependency Injection: Services are used in angular to share data and behavior across components. Dependency Injection (DI) is a design approach in which a class requests external sources for dependencies instead of making them itself. The DI system in Angular gives components and services their dependencies.

Routing: The Angular Router makes users switch from one view to another as they perform application tasks. It facilitates the creation of Single Page Applications (SPA), which are not required to load a new HTML page from the server to switch between views.

Observables and Reactive Programming: RxJS, a library for reactive programming, is extensively utilized by Angular. This enables the efficient handling of asynchronous operations, management of data flows, and propagation of changes using Observables, Observers, and Operators.

3.4. Backend Architecture

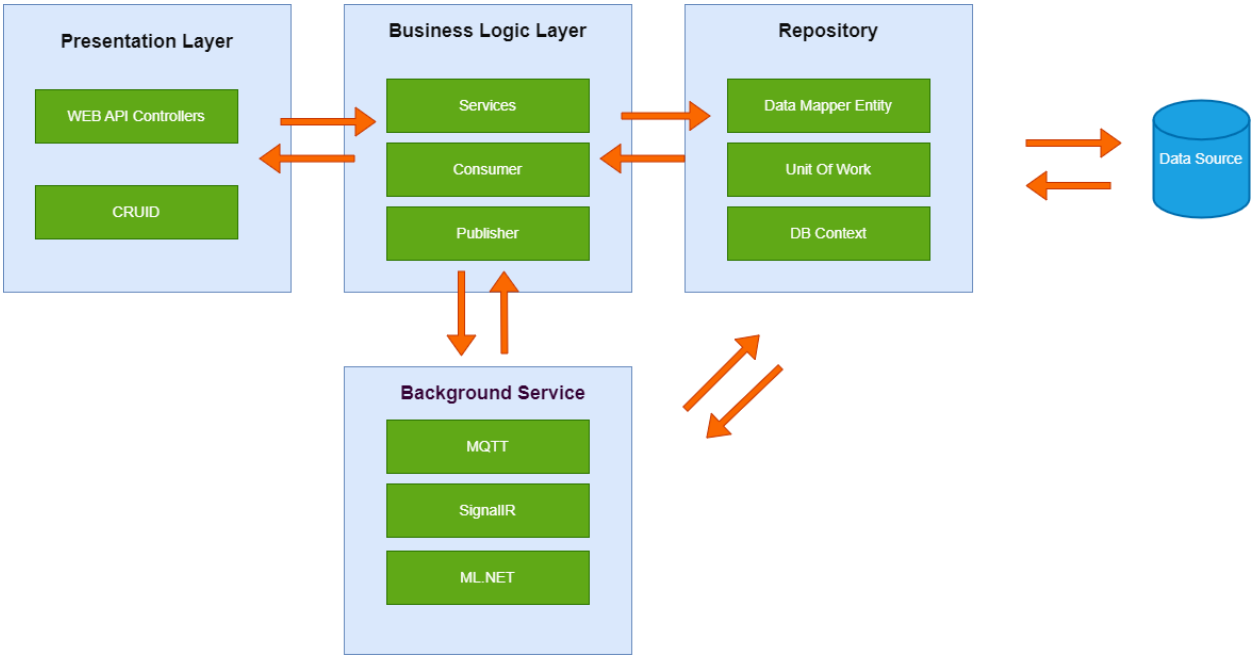


Figure 3.3: Backend Architecture

The backend application will be implemented as a Layered architecture. Layered architecture, known as n-tier architecture, is the common standard for most Java EE applications and is widely used in.NET applications. It assists in reducing the complexity of software into manageable chunks. Each layer is responsible for its own tasks, and each layer depends solely on the layer directly beneath it. Here is a breakdown of the typical layers and their responsibilities:

Presentation Layer (API Controllers):

- Presentation layer is the entry point where the front-end or external services will communicate with the backend. All the web API endpoints will be implemented in this layer.

Business Logic Layer (Service Layer):

- Business logic, computations, validations, and application-specific functionality.
- Acts as an intermediary between the Presentation and the Data Access layers.

Data Access Layer (DAL):

Responsible for accessing data from database. • Contains code for accessing data and maps this data to the business entities. This mapping can be done using ORMs (Object-Relational Mapping) tools. Entity framework will be used as ORM in the system.

Repository Pattern

The Layered architecture will be implemented based on the repository pattern. The repository pattern is a well-known design pattern used in application software architecture, especially in the context of domain-driven design. The main goal of this pattern is to separate the business logic of a program from the logic for accessing data. Repository pattern makes the system more maintainable, testable, and scalable by protecting the business logic from any knowledge of the underlying data sources. In the solution, data access layer will be used to maintain all the repositories.

Unit of Work

Unit of Work is a design pattern introduced with the Repository pattern to handle transaction administration in a domain-driven design context. The primary objective of the unit of work pattern is to ensure that multiple database operations occur within a single transaction boundary. The unit of work provides mechanisms for committing or reverting database changes. If a single operation within the unit of work fails, the entire sequence of operations can be reverted to ensure data consistency.

Background Layer

An additional layer will be added to the backend architecture to handle real-time communication with IoT devices, HTTP socket communication and machine learning integration. Background layer will interact with repository layer and service layers to perform background tasks. When an IoT device sends data, it can first be pushed to a message queue, ensuring that data processing is fast and does not block the device. Background workers can then process data from the queue.

3.5. UML Diagrams

3.5.1. Use Case Diagram

Use case diagrams help illustrate the functional requirements of a system from the user's perspective. It illustrates the interactions between different actors (users or external systems) and the system's use cases (specific functionalities). Use case diagrams are helpful for communicating and understanding system functionality at a high level. They provide a clear explanation of how various actors interact with the system and the actions they can take.

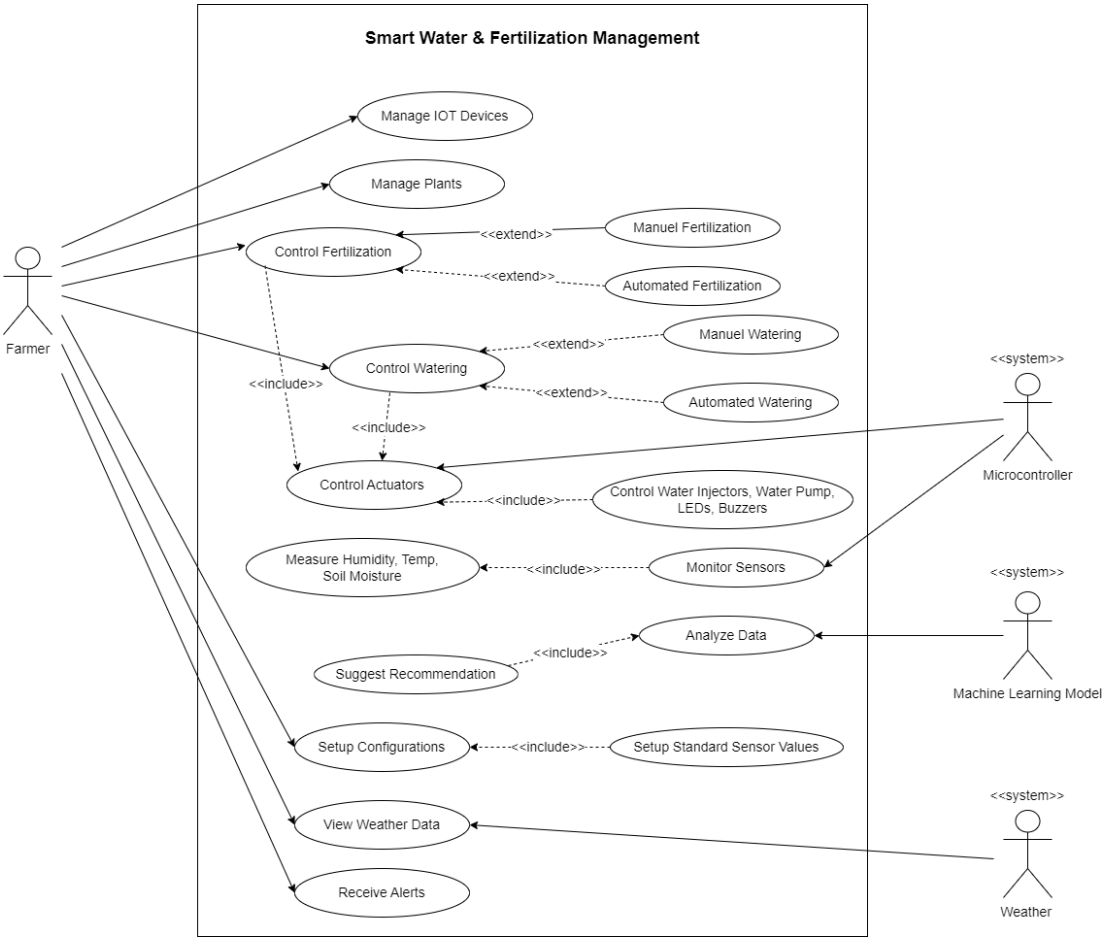


Figure 3.4: Use Case Diagram

As per the use case diagram in figure 3.4, the farmer is considered as the primary actor, and the microcontroller, weather service, and machine learning model are considered as external systems. Fertilization and watering processes can be managed manually by the farmer, or each plant can be set up for an automated procedure. The machine learning model is informed of the completed task at the end of each procedure. To obtain weather information, the system will integrate an external weather service. User types and user narratives as follows:

User Types

Table 3.1 consist the user types that will be used in the system.

Actor	Description
Farmer	The individual or entity responsible for managing the farm and using the system.
Microcontroller	A device that reads data from various sensors and sends commands to control systems.
Machine Learning Model	A software component that analyzes data and makes recommendations based on historical and current data.
Weather Station	A system that provides real-time weather data, which can be used to make informed decisions.

Table 3.1: User Types

Use Case Narratives

The following case narratives will be used in the system.

Use Case 1: Manage Plants

Attribute	**Description**
Use Case Name	Manage Plants
Primary Actor	Farmer
Precondition	Farmer has access to the plant management system. Plants are planted in the farm.
Postcondition	Plants are monitored, cared for, or updated as per the farmer's requirements.
Main Flow	1. Farmer navigate to plant management page. 2. Farmer selects the type of plants to manage. 3. Farmer can add new plants or Farmer can views the status and health of the already added plants and update the care settings for the plants, if necessary 4. Farmer sends the updated settings to the system. 5. The system validates the settings and applies them to the plants. 6. The system confirms to the Farmer that the settings have been successfully updated.
Extensions	If the care settings are invalid: The system shows an error message. The Farmer corrects the settings and retries. If the system is not responding: The system shows an error message. The Farmer checks the system and retries.
Special Requirements	The system should provide real-time feedback on the status of the plants. The system should be able to integrate with IoT devices for automated care of plants.

Use Case 2 : IOT Device Manage

Attribute	Description
Use Case Name	IOT Device Manage
Primary Actor	Farmer
Precondition	Farmer has access to the IoT device management system. IoT devices are installed in the farm.
Postcondition	IoT devices are configured, monitored, or updated as per the farmer's requirements.
Main Flow	1. Farmer navigate to IOT device management page. 2. Farmer selects the IoT devices to manage. 3. Farmer views the status and configuration of the selected devices. 4. Farmer updates the configuration settings for the devices, if necessary. 5. Farmer sends the updated configuration settings to the system. 6. The system validates the settings and applies them to the devices. 7. The system confirms to the Farmer that the settings have been successfully updated.
Extensions	If the configuration settings are invalid: The system shows an error message. The Farmer corrects the settings and retries If the system is not responding: The system shows an error message. The Farmer checks the system and retries.
Special Requirements	The system should provide real-time feedback on the status of the IoT devices. The system should be able to integrate with various types of IoT devices.

Use Case 3: Control Fertilization

Attribute	Description
Use Case Name	Control Fertilization
Primary Actor	Farmer
Stakeholders and Interests	Fertilization System
Precondition	The fertilization system is installed and operational.
Postcondition	Fertilization is controlled as per the farmer's input or automated settings. Fertilization data is saved.
Main Flow	1. The Farmer decides to control the fertilization process. 2. The System provides options for manual or automated control. 3. The Farmer selects the desired option. 4. The System controls the fertilization process as per the selected option.
Extensions	1. If the Farmer chooses Manual Control The System triggers the "Manual Fertilization" use case (extend relationship). 2. If the Farmer chooses Automated Control The System triggers the "Automated Fertilization" use case (extend relationship).
Special Requirements	The System must respond to the Farmer's input within 5 seconds.

Use Case 4 : Control Watering

Attribute	Description
Use Case Name	Control Watering
Primary Actor	Farmer
Stakeholders and Interests	Watering system
Precondition	The watering system is installed and operational.
Postcondition	Watering is controlled as per the farmer's input or automated settings. Watering data is saved.
Main Flow	1. The Farmer decides to control the water process. 2. The System provides options for manual or automated control. 3. The Farmer selects the desired option. 4. The System controls the water process as per the selected option.
Extensions	1. If the Farmer chooses Manual Control The System triggers the "Manual Watering" use case (extend relationship). 2. If the Farmer chooses Automated Control The System triggers the "Automated Watering" use case (extend relationship).
Special Requirements	The System must respond to the Farmer's input within 5 seconds.

Use Case 5 : Control Actuators

Attribute	Description
Use Case Name	Control Actuators
Primary Actor	Microcontroller
Precondition	Microcontroller is operational and connected to actuators; Commands are received from the Farmer or System.
Postcondition	Actuators are controlled as per the received commands; Actuator status is logged.
Main Flow	1. The Microcontroller receives a command to control actuators. 2. Microcontroller validates the received command. 3. The Microcontroller initiates the corresponding control actions for the actuators (include relationships: Control Relays, Water Pump, Fertilizer Pumps. 4. The Actuators execute the commands and change their states. 5. The Microcontroller confirms successful execution and logs the actuator status.
Extensions	If the received command is invalid: The Microcontroller logs an error and ignores the command. If any actuator is offline or not responding: The Microcontroller logs an error and may trigger an alert
Special Requirements	The Microcontroller must validate commands within 2 seconds. The Microcontroller must securely authenticate the source of the commands. The Microcontroller must log all actuator control activities for auditing.

Use Case 5 : Monitor Sensors

Attribute	Description
Use Case Name	Monitor Sensors
Primary Actor	Microcontroller
Precondition	Microcontroller is operational and connected to sensors; Sensors are calibrated and operational.
Postcondition	Sensor data is collected, processed, and logged; Alerts are generated if necessary.
Main Flow	1. The Microcontroller initiates the sensor monitoring cycle. 2. The Microcontroller sends commands to sensors to start measurements (include relationships: Measure Humidity, Temp, Soil Moisture). 3. The Sensors take measurements and send the data back to the Microcontroller. 4. The Microcontroller processes and logs the received sensor data. 5. The Microcontroller checks if the sensor data is within acceptable ranges and decides whether alerts are necessary. 6. The Microcontroller ends the sensor monitoring cycle.
Extensions	If any sensor is offline or not responding. The Microcontroller logs an error and may trigger an alert. If the sensor data is outside acceptable ranges. The Microcontroller triggers an alert and may initiate corrective actions (e.g., adjusting actuators).
Special Requirements	The Microcontroller must validate sensor data within a defined time interval. The Microcontroller must securely authenticate the source of the sensor data. Microcontroller must log all sensor monitoring activities for auditing. The system must be capable of alerting the Farmer in real-time if critical thresholds are breached.

Use Case 6: Analyze Data

Attribute	Description
Use Case Name	Analyze Data
Primary Actor	Machine Learning Model
Precondition	The Machine Learning Model is trained and operational. Sensor data is available and valid.
Postcondition	Analysis is completed; Recommendations are generated and communicated to the Farmer.
Main Flow	1. The Machine Learning Model initiates the data analysis process. 2. The Model retrieves the latest sensor data from the data repository. 3. The Model processes and analyzes the retrieved data. 4. The Model includes the "Suggest Recommendation" use case to generate actionable insights based on the analyzed data. 5. The Model logs the analysis results and generates recommendations. 6. The Model communicates the recommendations to the Farmer through a user interface or notification system. 7. The Model ends the data analysis process.
Extensions	If the retrieved sensor data is incomplete or invalid: The Model logs an error and may trigger an alert. If the Model cannot generate recommendations: The Model logs an error and may trigger an alert to the Farmer.
Special Requirements	The Model must validate the integrity and validity of the sensor data. The Model must log all data analysis activities for auditing. The system must be capable of alerting the Farmer in real-time if critical issues are detected. The recommendations must be generated within a defined time interval.

Use Case 8 : Setup Configurations

Attribute	Description
Use Case Name	Setup Configurations
Primary Actor	Farmer
Precondition	System is working based on configurations.
Postcondition	The configurations are updated and saved. The system operates based on the new configurations.
Main Flow	1. The Farmer initiates the configuration setup process. 2. The Farmer includes the "Setup Standard Sensor Values" use case to define and update standard values for various sensors. 3. The Farmer reviews and modifies other configurations (e.g., alert thresholds, irrigation schedules). 4. The Farmer submits the updated configurations. 5. The system validates the submitted configurations. 6. The system saves the validated configurations. 7. The system confirms to the Farmer that the configurations have been successfully updated. 8. The Farmer ends the configuration setup process.
Extensions	If the submitted configurations are invalid: The system displays an error message to the Farmer. The Farmer can correct the configurations and resubmit.
Special Requirements	System should validate the configurations. Configuration should be updated without getting errors in current flow.

Use Case 9 : View Weather Data

Attribute	Description
Use Case Name	View Weather Data
Primary Actor	Farmer
Precondition	The Weather Station is operational and sending data. The Farmer is authenticated and authorized to view weather data.
Postcondition	The Farmer has viewed the current weather data.
Main Flow	1. The system appears the weather data in dashboard to the Farmer. 2. The Farmer reviews the weather data.
Extensions	If the system cannot retrieve the weather data: The system displays an error message to the Farmer.
Special Requirements	The system must ensure the security and privacy of the weather data. The system must update the weather data at regular intervals. The system must display the weather data in a user-friendly format. The system must log all access to the weather data for auditing. The system must be capable of handling different types of weather data (e.g., temperature, humidity, wind speed).

Use Case 10 : Receive Alerts

Attribute	Description
Use Case Name	Receive Alerts
Primary Actor	Farmer
Precondition	The alerting system is operational. The Farmer is registered to receive alerts.
Postcondition	The Farmer has received and acknowledged the alert.
Main Flow	1. A critical condition (e.g., extreme weather, low soil moisture) triggers an alert in the system. 2. The Microcontroller or Weather Station sends an alert to the system. 3. The system processes the alert and determines that it should be sent to the Farmer. 4. The system sends the alert to the Farmer via the preferred communication channel (e.g., SMS, email, app notification). 5. The Farmer receives and reviews the alert. 6. The Farmer acknowledges the receipt of the alert through the system.
Extensions	If the system cannot send the alert to the Farmer. The system logs the failure. The system retries sending the alert after a predefined time interval. If the alert is still not sent after several attempts, the system escalates the issue to a system administrator.
Special Requirements	The system must ensure the security and privacy of the alerts. The alerts must be sent in real-time or near-real-time. The system must log all sent alerts for auditing. The alerts must be clear, concise, and actionable. The system must allow the Farmer to customize alert preferences.

3.5.2. ER Diagram

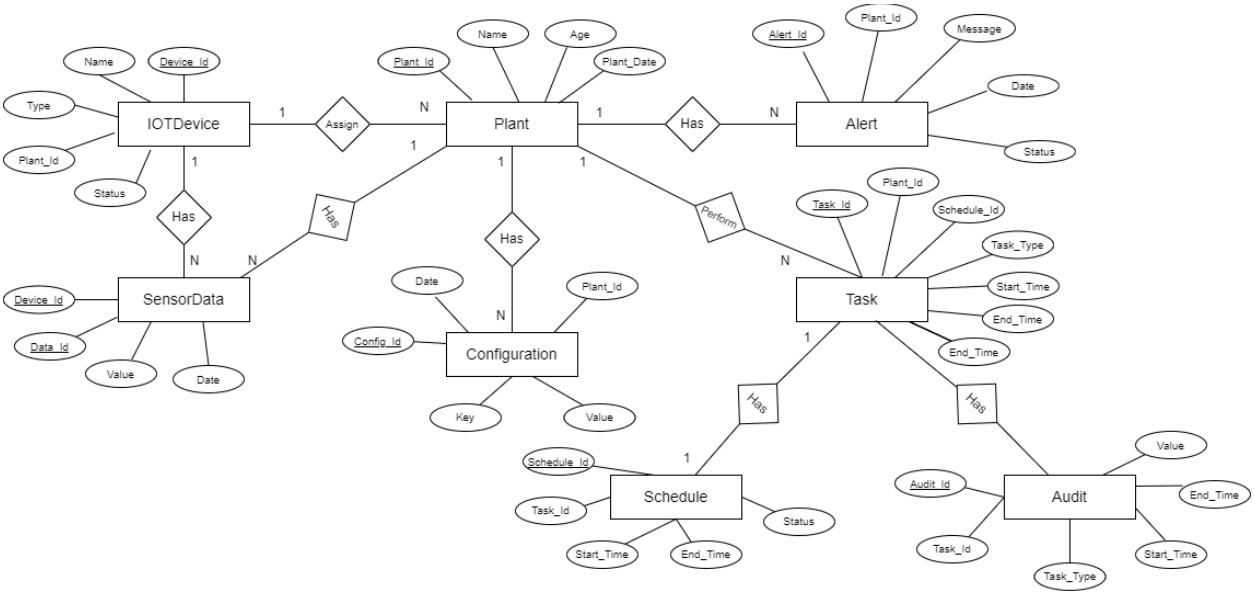


Figure 3.5:Entity Relationship Diagram

As per the Figure 3.5, Each plant will be associated with multiple tasks, and there will be a predefined schedule for each task. Fertilization and watering processes will be executed through the tasks and schedules. Plants can have multiple sensors and actuators connected, but sensors and actuators can only be attached to one plant. Audit details of each task will be inserted into a separate table for the machine learning process.

3.5.3. Sequence Diagrams

Sequence diagrams describe the interaction between the objects in sequence order that those interactions occur. Sequence diagram organized based on time with the time flow running from top to bottom of the diagram. The following figures describe main flows of the system.

Scenario 1: Plant management

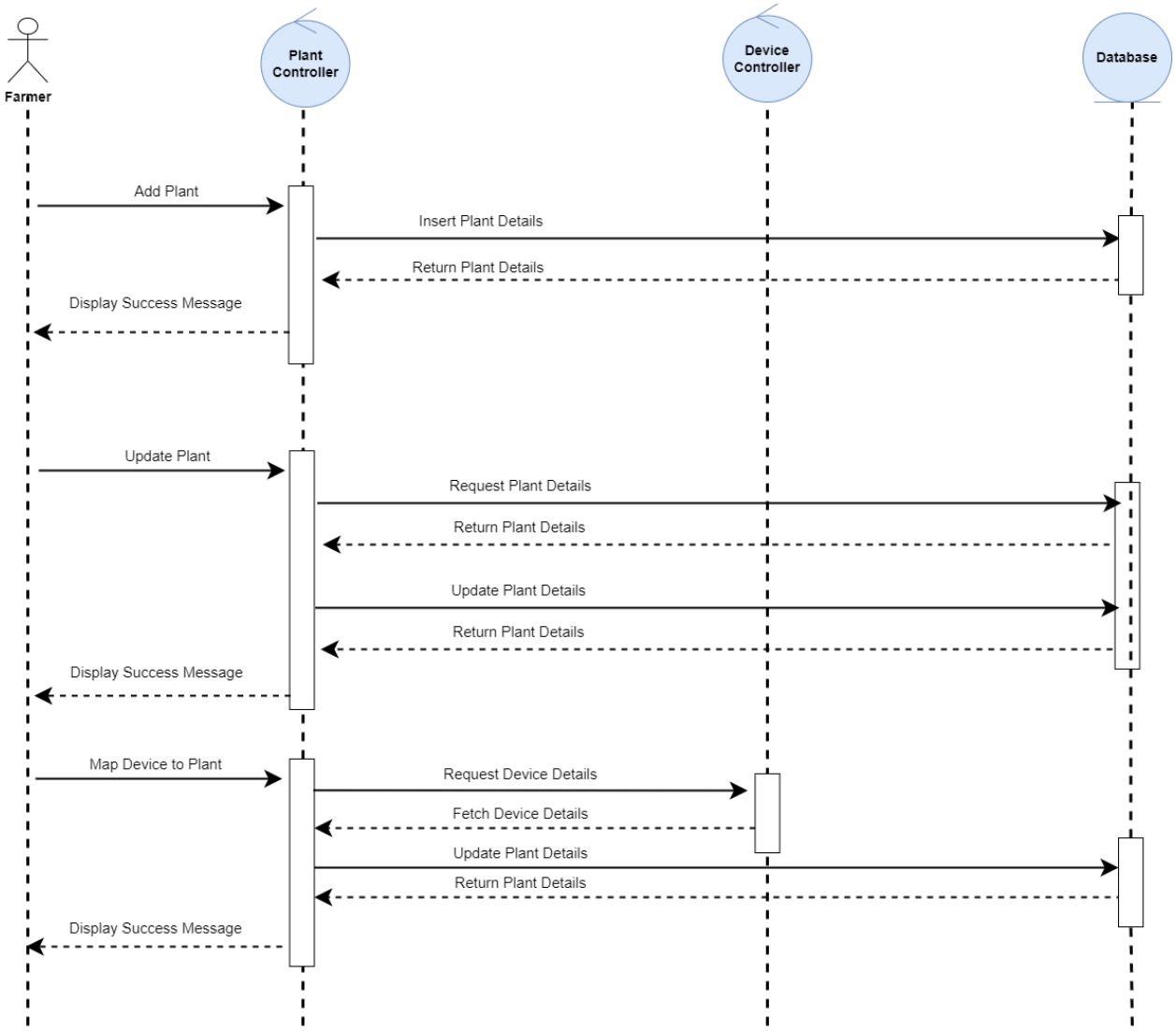


Figure 3.6:Sequence diagram for plant management

Figure 3.6 describes plant insert, plant details update and plant mapping to devices. User can map devices to plant at the initial plant insertion or after the plant creation.

Scenario 2: Device Management

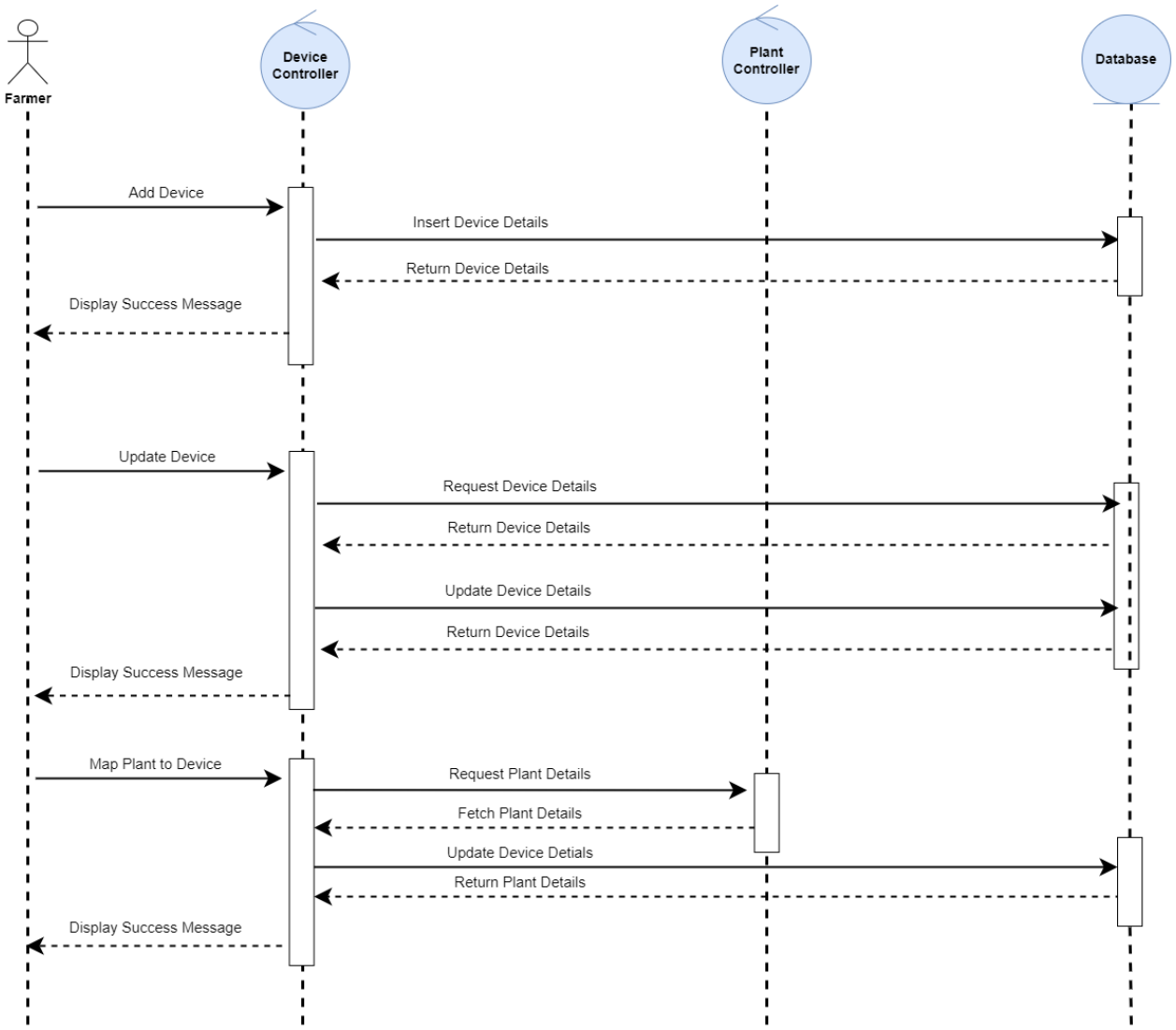


Figure 3.7: Sequence diagram for device management

Figure 3.7 describes device insert, device details update and device mapping to plant. At this point one device can only be assigned to one plant.

Scenario 3: Configuration Management

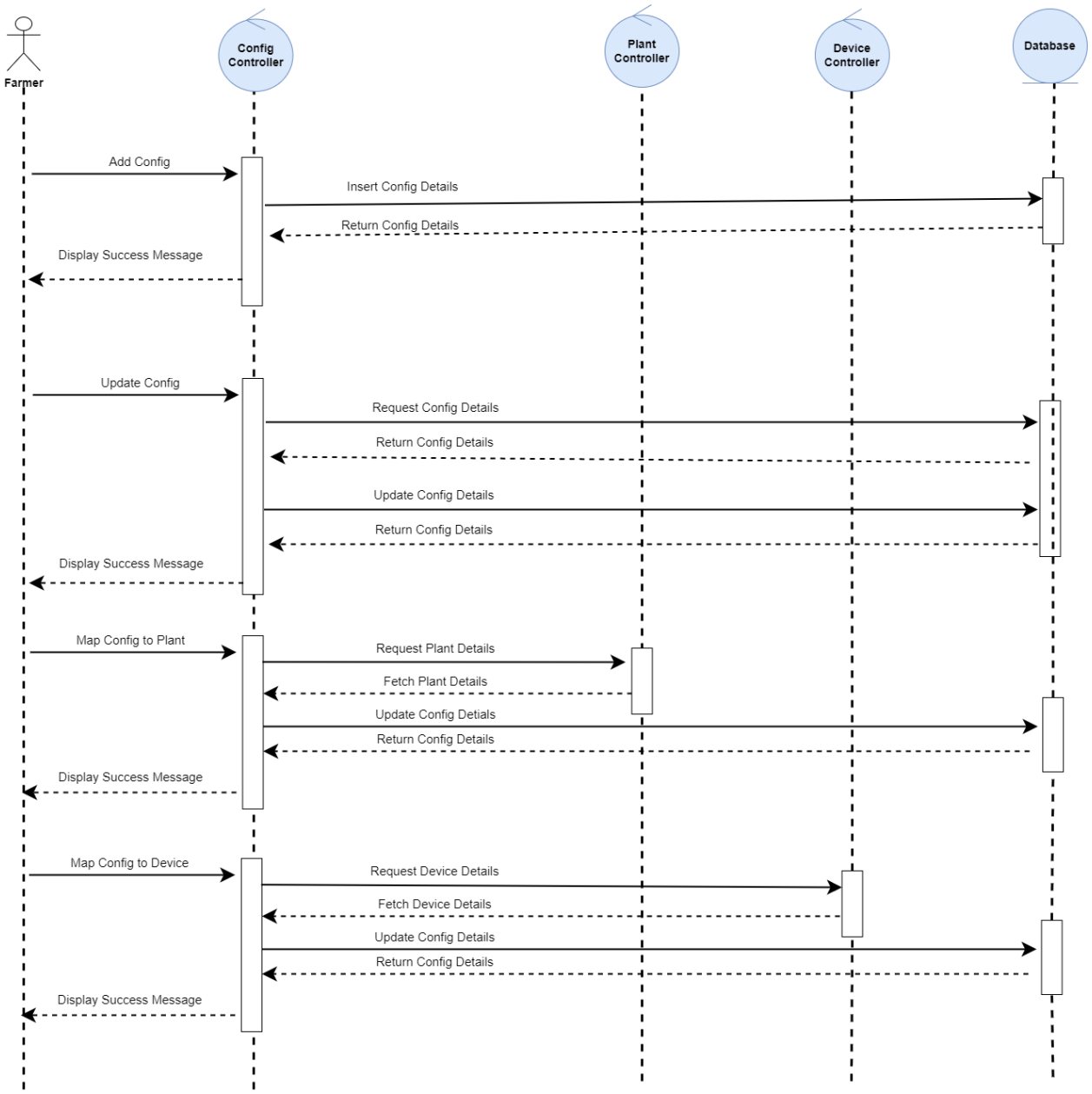


Figure 3.8: Sequence diagram for configuration management

Figure 3.8 describes configuration insert, configuration details update, and configuration mapping to plants and devices. Configuration is common for both plants and devices; it stores a value against the specified key.

Scenario 4: Watering Management

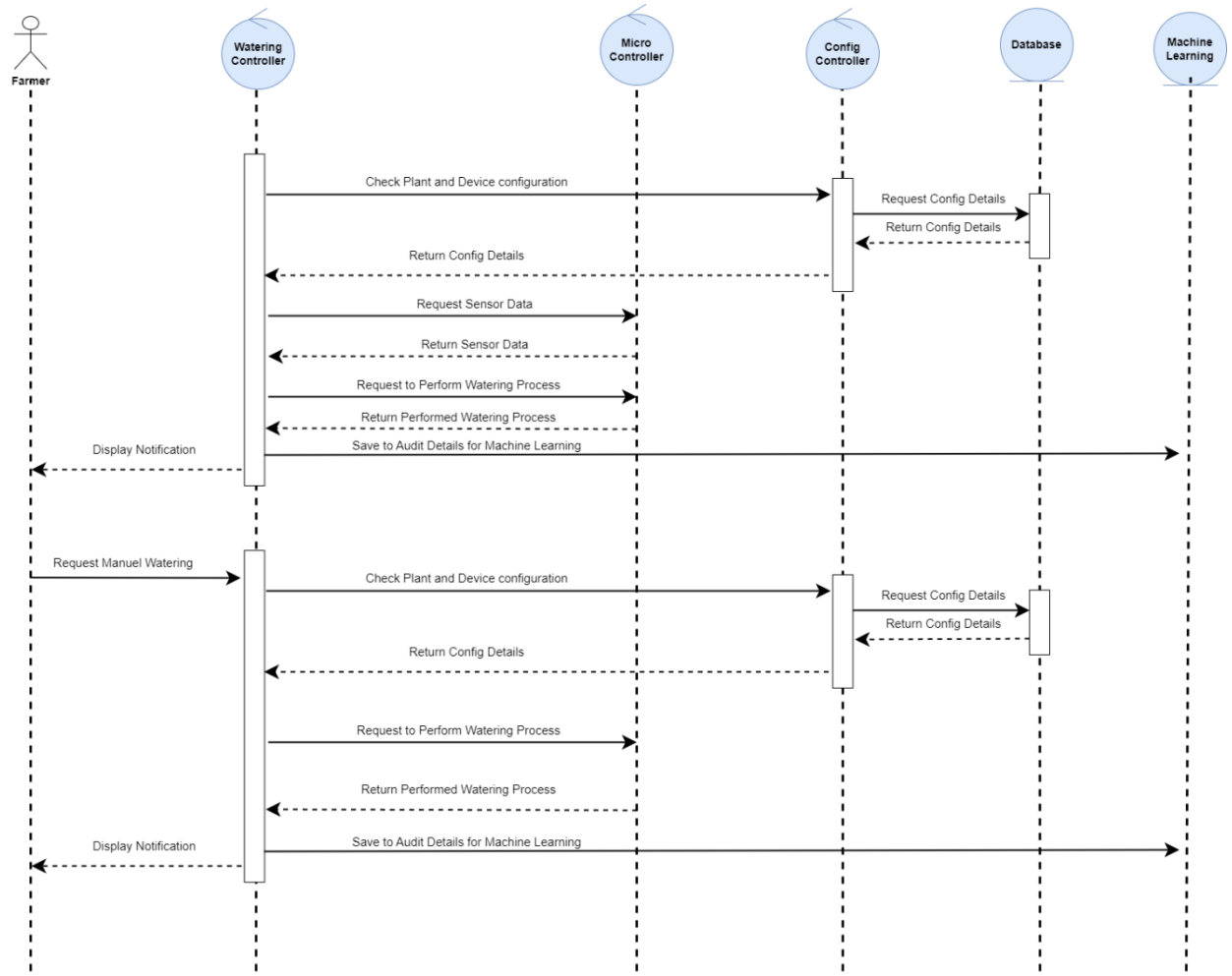


Figure 3.9: Sequence diagram for watering management

Figure 3.9 describes one of the main flows of the system. Watering process can be performed based on user involvement, or it can be set up to automate based on sensor values and default config values.

Scenario 5: Fertilization Management

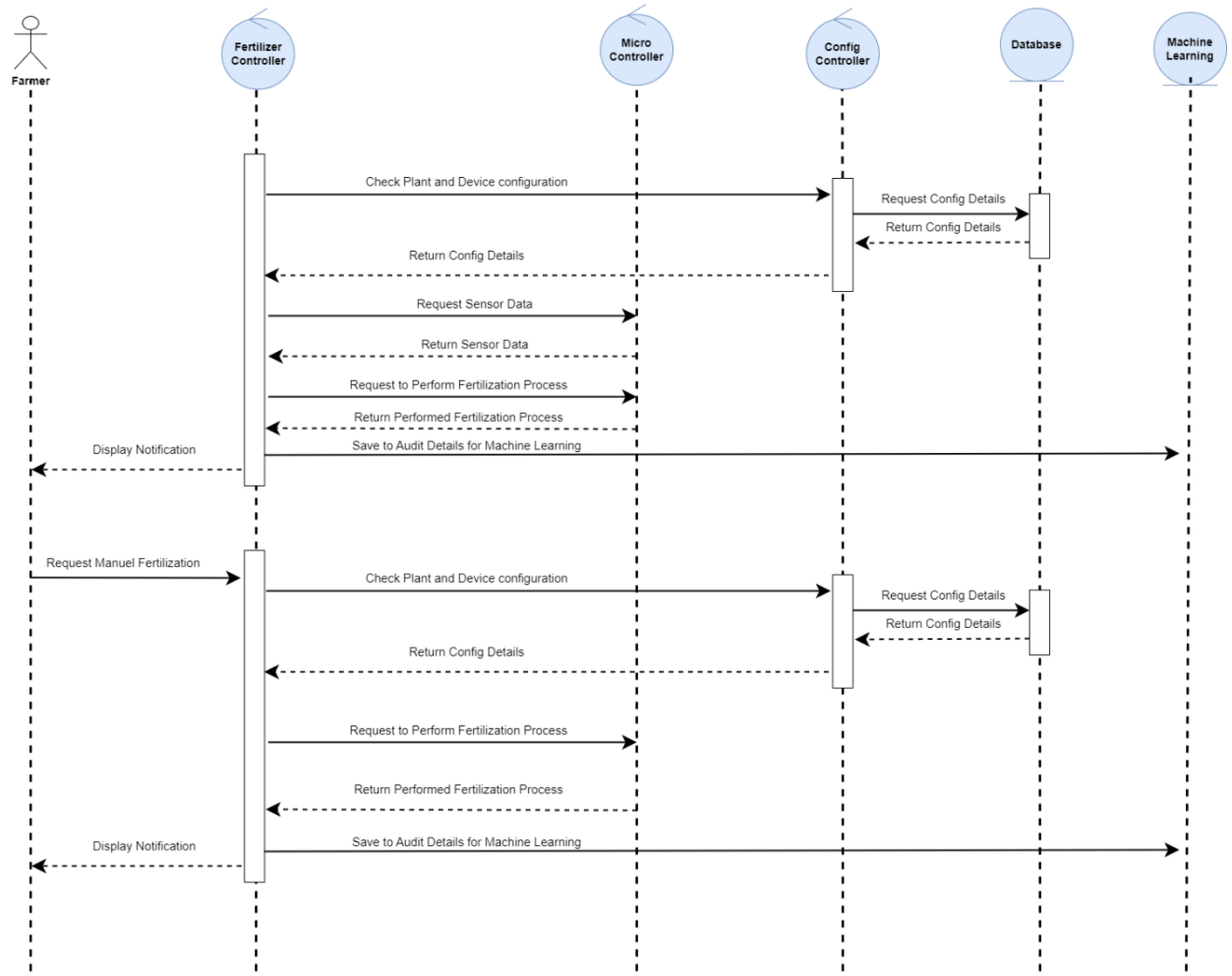


Figure 3.10: Sequence diagram for fertilization management

Figure 3.10 describes one of the main flows of the system. The fertilization process can be performed based on user involvement, or it can be set up to automate based on sensor values and default configuration values.

3.5.4. Class Diagram

The following class diagram illustrates the system's structure in terms of classes, their attributes, methods, and object relationships. Class diagrams are typically used to depict the static view of a system.

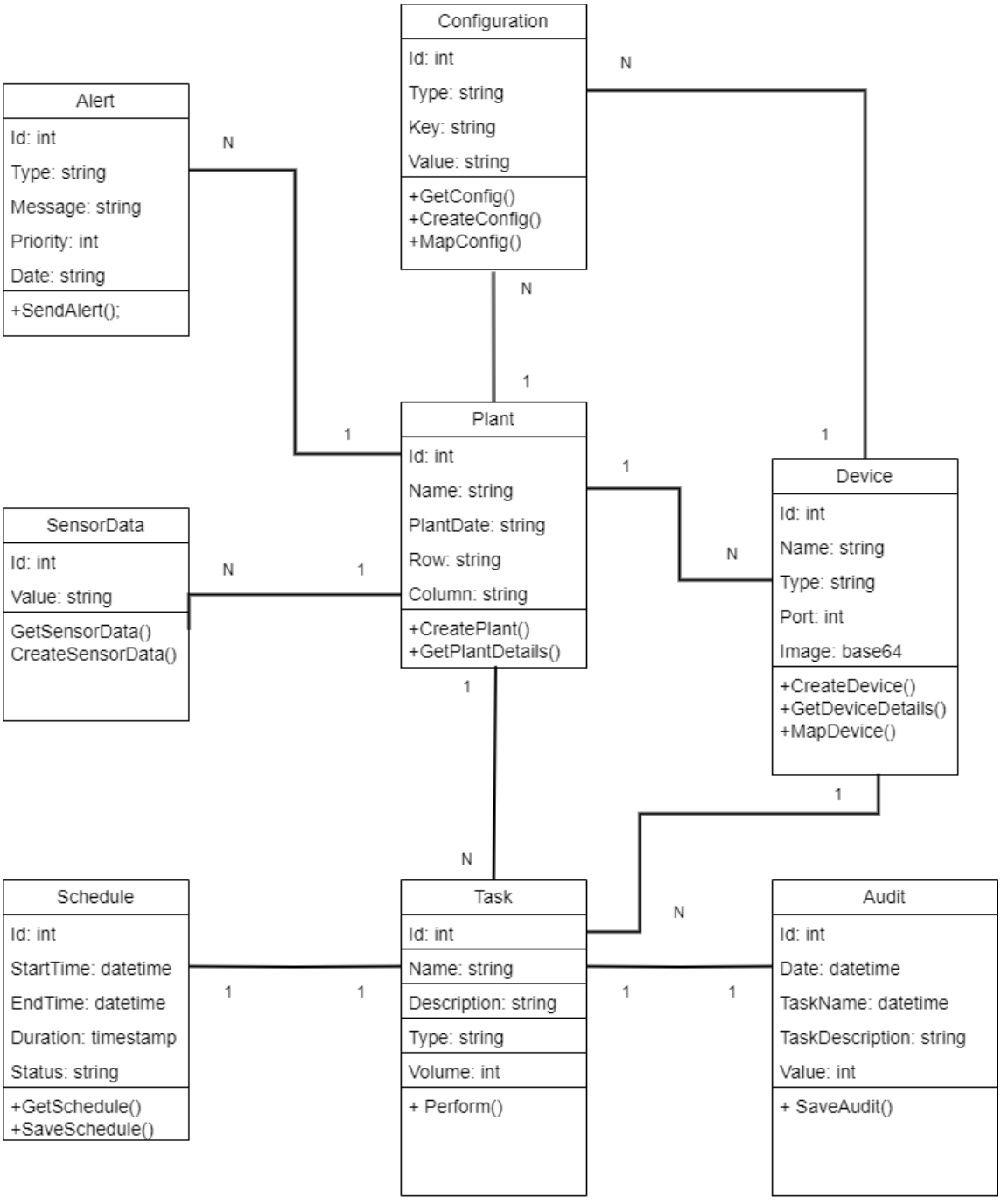


Figure 3.11: Class Diagram

3.6. User Interface Design

User Interface mainly focuses on appearance, feel, and interactivity in the system. It aims to provide a simple, effective, and user-friendly experience by serving as the link between the user's experience and the system's functionality. This section describes the visual elements, layout structures, and interactions that will be implemented to create a consistent user experience.

3.6.1. UI Colors

User Interface color scheme consists primarily of various shades of green, enhanced by neutral tones to create a balanced and visually attractive user interface. Since the system is related to agriculture and environmental behavior, the use of green shades make it feel calm and natural, which makes it easier for the user to navigate through the system.

3.6.2. System Layouts

The system style uses rounded shapes for things like buttons and layouts, as well as green colors. The design feels softer and kinder because the corners are rounded, which matches the system's natural theme. System UX design has mainly focus to achieve web application feel and stylish appearance. There will be two main dashboards for the system.

- Main Dashboard – User can view all plant information on this dashboard.
- Plant dashboard – User can view and manage each plant.

3.6.3. Main Dashboard

The Figure 3.12 is main dashboard design and it contains multiple sections including header, weather card, plant grid, plant card, task bar.

- **Header:** The header section is common for all pages, and it contains all main pages in the system.
- **Weather Card:** This section contains all the information related to the weather conditions in the location.
- **Plant Grid:** All plants will be appeared in plant grid. Users can view information related to the plant by selecting each grid item.
- **Plant Card:** Whenever a user selects the plant grid item, information relevant to the selected plant will be displayed in the plant card.
- **Task Bar:** Task progress will be visible to user with the task bar. It will be displayed the progress of each task.

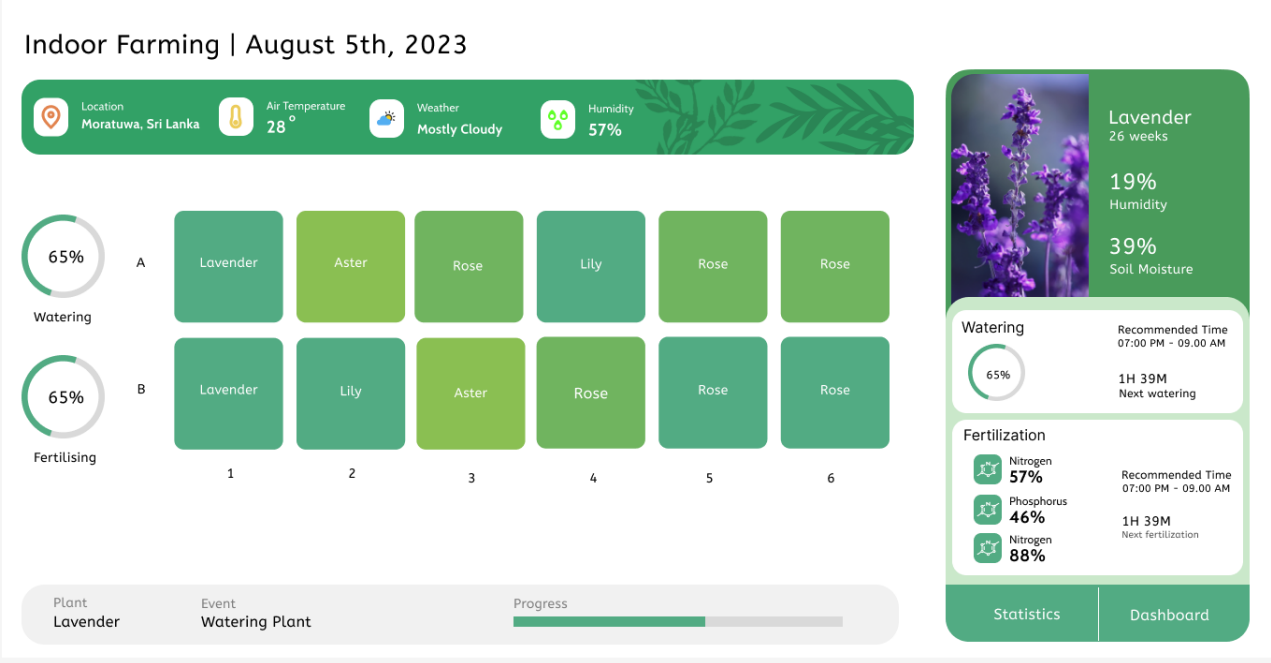


Figure 3.12:Main Dashboard

3.6.4. Plant Dashboard

Figure 3.13 is the design of plant dashboard and it is including all watering and fertilization options. The plant dashboard shows all the information about the selected plant, and the user may check all sensor details as well as control the watering and fertilization processes. Except for the plant grid, all components of the main dashboard are available in the plant dashboard. Two major parts will be introduced to control the system's key functions.

- **Watering Control** – View related sensor details, control watering process, view watering related recommendations.
- **Fertilization Control** - View related sensor details, control fertilization process, view fertilization related recommendations.

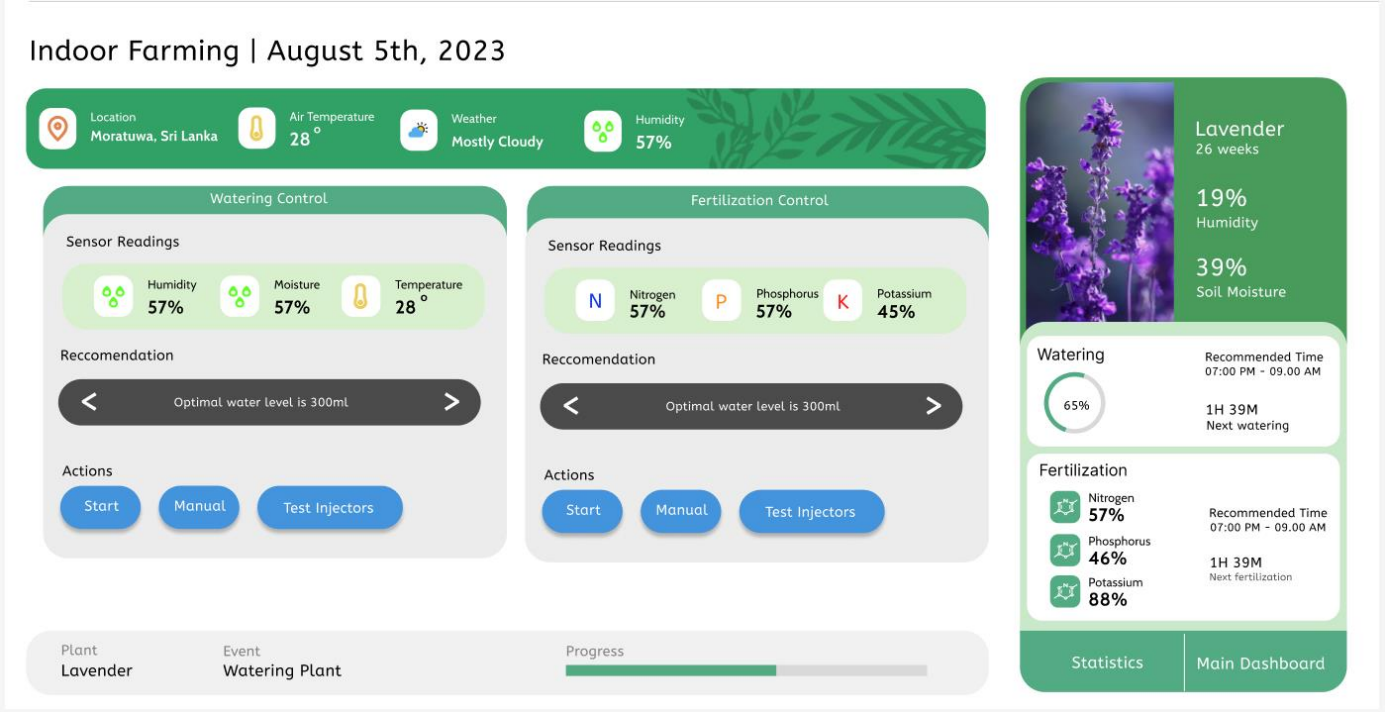


Figure 3.13 : Plant Dashboard

3.6.5. Popups

Pop-up windows will provide access to features such as managing plants, devices, and setups. These pop-ups are user-friendly and intuitive, allowing for quick and simple navigation. They will appear when you click on specific buttons or options, allowing you to have a seamless experience without leaving the current web page. This method aims to make the system more efficient and user-friendly. Figure 3.14 and Figure 3.15 are the designs of the Popup components.

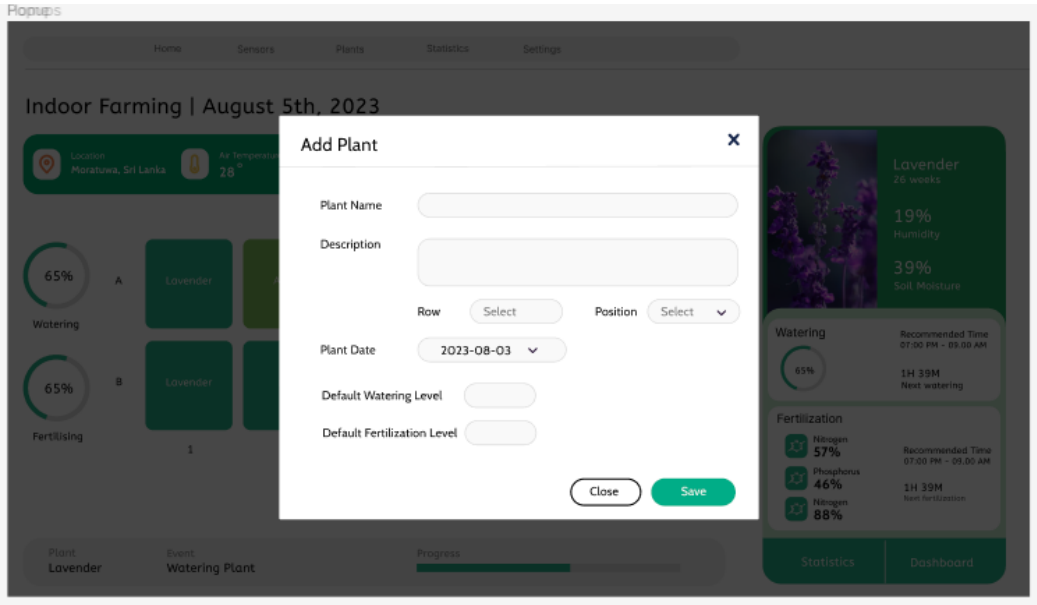


Figure 3.14: Add plant popup

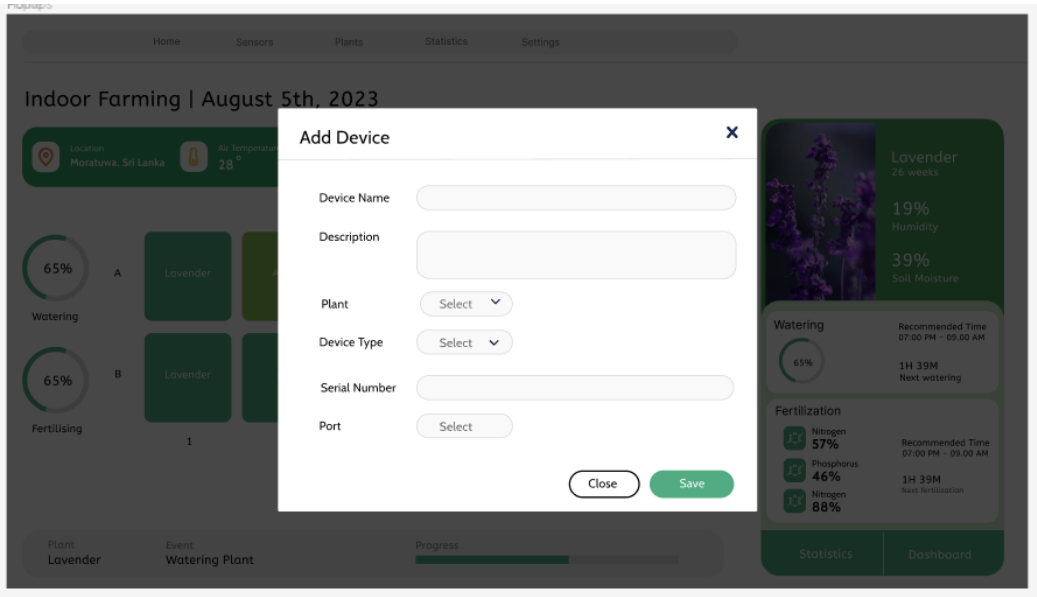


Figure 3.15: Add device popup

4 CHAPTER - IMPLEMENTATION

This chapter describes the implementation of the system including software and hardware components, related technologies, system design and architecture, hardware components, software components and system deployment will be focus in this chapter.

4.1. Related Technologies

The proposed smart water and fertilize system consist both software system and hardware setup that communicate each other over wireless connection. Software system has developed as a web-based application that consist both frontend and backend. Following latest technologies have used to develop the system with industry standard and the well-known best practices.

4.1.1. Frontend Technologies

UI and the frontend application has Implemented with Angular. Angular is a popular open-source web application based on JavaScript which mostly use for single page applications that update HTML pages dynamically. Angular is build using Typescript and entire frontend application coded with typescripts that's makes easier to write more complex code and maintain large scale applications.

UI components implemented with HTML, SCSS and Bootstrap 5 styles used to make components more attractive. Ngx-boostrap used to integrate few UI components.

SignalR Angular and Redux state management tools has utilized for real-time feature in the system.

4.1.2. Backend Technologies

Backend is implemented with .Net Core, developed by Microsoft. It's a free open-source, cross platform framework used for building complex applications. .NET core build on the top of C# language. Entity Framework (EF) has integrated with .NET Core as object relational mapping (ORM) framework to simplify data operations. EF allows to map domain entities to database tables, without using most of data-access code. PostgreSQL database used with Entity Framework to store data of the system. PostgreSQL is powerful open-source object relational database that provide more reliability, data integrity.

SignalR technology used to facilitates real-time communication. SignalR uses WebSockets to notify change of events from server to connected clients real time.

MQTT messaging protocol used to communicate between system backend and hardware components. MQTT is lightweight and efficient messaging protocol suitable for environment that has limited bandwidth.

4.2. Hardware Implementation

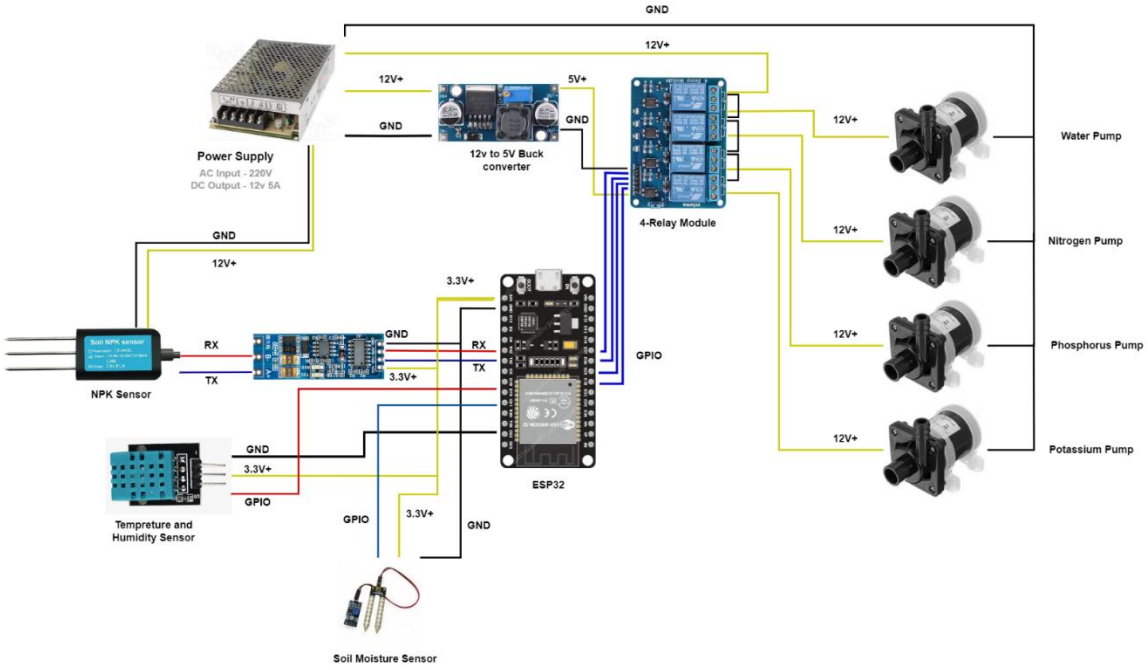


Figure 4.1:Hardware Components Diagram

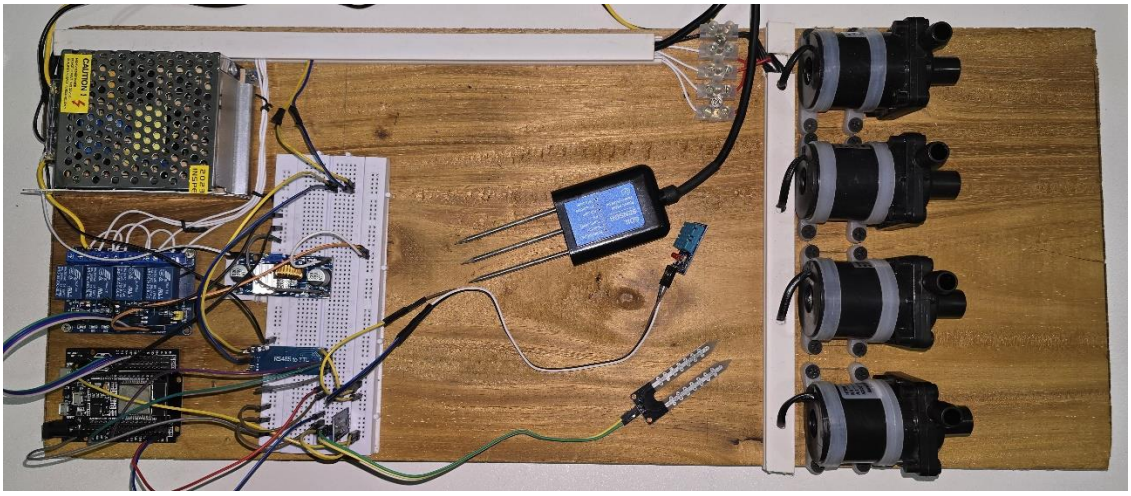


Figure 4.2: Actual Hardware Implementation

The connection between electronic components is illustrated in Figure 4.1. Actual hardware configuration implementation is depicted in Figure 4.2.

4.2.1. Hardware Components

ESP32 Devkit 1 Microcontroller: The ESP32 is WIFI and Bluetooth inbuilt microcontroller which is operate in 3.3V. This development board consume low power and consist dual-core processor and 512KB memory. ESP32 board provides multiple GPIOs including serial communication ports. It is capable to operate in USB power or external 5v power input.

Water Pump: DC 12v Powered water pump with brush less magnetic.

4-Relay Module: A relay module used to control high voltage devices which connected microcontroller. This module consist 4 relays and it allow use control 4 different devices from microcontroller. This module operates on 5v.

12v Buck Convertor: Buck convertor is electronic circuit that convert higher voltage to lower voltage. This module is capable to convert 12v to 5-11v range.

RS485 to TTL convertor: Signal conversion circuit that convert RS485 to TTL signals in serial communication.

NPK Sensor: NPK sensor is specialized device used in agriculture industry to measure the nutrients levels of Nitrogen, Phosphorus and Potassium in soil. This device operates in 5-12v range and support only serial communication in broad rate of 4800.

Temperature and Humidity Sensor: DHT11, which is a widely used temperature and humidity sensor that can measure temperature range from 0°C to 50°C with a ±2°C accuracy. DHT11 has a humidity measurement range of 20% to 90% RH (Relative Humidity) with a ±5% RH accuracy. This device operates at 3.3V and provide digital signals only.

Soil Moisture Sensor: Device used to measure the water content in soil. This device operates at 3.3V and provide both analog and digital signals.

4.2.2. Sensor Integration

According to the Figure 4.1, three sensors are used to measure nutrient levels, soil properties, and environmental variables. The DHT11 and Soil moisture sensors are directly attached to the ESP32 board as they both operate on a 3.3v power supply, which can be provided by the microcontroller. Two digital input pins are utilized for the purpose of reading sensor information.

The NPK sensor requires a power supply of 5-12V for its operation and is connected to an external power source that generates 12V 5A from a 220V AC input. The NPK sensor only supports serial communication. The NPK sensor generates RS485(TIA-485(-A)) signal, which must be converted to TTL (Transistor-Transistor Logic) in order to be recognized by the microcontroller. Thus, an RS485 to TTL converter is linked between the NPK sensor and the ESP32 microcontroller.

4.2.3. Actuator Integration

According to the Figure 4.1, there are four pumps linked to the system. All of these pumps are functioning at a voltage of 12 volts and require the use of relays to respond to incoming signals from a microcontroller. The pumps are linked to an external power source via the relay modules. The pump motor operates exclusively when a signal is received from the microcontroller through each relay. The 4-Relay module functions at a voltage of 5 volts, and a Buck converter is installed to convert an external power supply of 12 volts to the required 5 volts for the relay module. The ESP32 allocates four GPIO pins for the relay module.

4.2.4. Code Segments

The ESP32 microcontroller is programmed using the ArduinoIDE with ESP extensions. The following sections provide detailed explanations of the main code snippets.

WiFi Connection and MQTT connection

```
void setup() {
  // Set software serial baud to 115200;
  Serial.begin(115200);
  SerialPort.begin(4800, SERIAL_8N1, 16, 17);
  delay(10);
  // Connecting to a WiFi network
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }
  Serial.println("Connected to the WiFi network");
  //connecting to a mqtt broker
  client.setServer(mqtt_broker, mqtt_port);
  client.setCallback(callback);
  while (!client.connected()) {
    String client_id = "esp32-client-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public mqtt broker\n", client_id.c_str());
    if (client.connect(client_id.c_str())) {
      Serial.println("Public mqtt broker connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
  // subscribe to topic
  client.subscribe("SWFS/wateringtasks");
  client.subscribe("SWFS/fertlizetasks");
```

Figure 4.3: Wifi Connection Initiation

The program initially establishes a WiFi connection and a connection to a MQTT broker.

Perform Watering and Fertilizing tasks

After establishing a connection, the program will subscribe to two topics, namely "SWFS/wateringtasks" and "SWFS/fertilizetasks", in order to carry out tasks received from the .Net backend. Currently, the callback method has been registered with the MQTT client. When a message is received, it will be passed to each method as Figure 4.4.

```
void callback(char *topic, byte *payload, unsigned int length) {
  Serial.print("Message arrived in topic: ");
  Serial.println(topic);
  Serial.print("Message:");
  String messageTemp;
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
    messageTemp += (char)payload[i];
  }
  Serial.println();
  Serial.println("-----------------------");

  if (strcmp(topic, "SWFS/wateringtasks") == 0) {
    Serial.println(topic);
    handleWateringRelay(messageTemp, length);
  }

  if (strcmp(topic, "SWFS/fertlizetasks") == 0) {
    Serial.println(topic);
    handleFertilizerRelay(messageTemp, length);
  }
}
```

Figure 4.4: MQTT Subscription

The watering and fertilization operation is carried out using following methods with the use of variables in the received JSON message. Essentially, the relays will receive power for a predetermined duration and subsequently deactivate.

```
void handleWateringRelay(String message, unsigned int length) {
  DynamicJsonDocument doc(512);
  deserializeJson(doc, message);

  if (doc.containsKey("Port") && doc.containsKey("Duration")) {
    TaskDto task;
    task.Port = doc["Port"];
    task.Duration = doc["Duration"];

    Serial.println("Watering Task Received:");
    Serial.print("Port: ");
    Serial.println(task.Port);
    Serial.print("Duration: ");
    Serial.println(task.Duration);

    // Control watering relay
    digitalWrite(task.Port, LOW);
    delay(task.Duration);
    digitalWrite(task.Port, HIGH);
  } else {
    Serial.println("Invalid JSON format for Watering Task");
  }
}
```

Figure 4.5: Watering Handle Method

```
void handleFertilizerRelay(String message, unsigned int length) {
  DynamicJsonDocument doc(1024);
  deserializeJson(doc, message);

  if (doc.is<JsonArray>()) {
    JsonArray tasks = doc.as<JsonArray>();
    Serial.println("Fertilizer Tasks Received:");
    for (const auto &task : tasks) {
      if (task.containsKey("Port") && task.containsKey("Duration")) {
        TaskDto fertilizerTask;
        fertilizerTask.Port = task["Port"];
        fertilizerTask.Duration = task["Duration"];

        Serial.print("Port: ");
        Serial.println(fertilizerTask.Port);
        Serial.print("Duration: ");
        Serial.println(fertilizerTask.Duration);

        // handle fertilizer relays
        activateRelay(fertilizerTask);
        digitalWrite(fertilizerTask.Port, LOW);
        delay(fertilizerTask.Duration);
        digitalWrite(fertilizerTask.Port, HIGH);
      } else {
        Serial.println("Invalid JSON format for a Fertilizer Task");
      }
    }
  } else {
    Serial.println("Invalid JSON format for Fertilizer Tasks");
  }
}
```

Figure 4.6: Fertilizing Handle Method

Obtaining Data from Sensors

The DHT humidity and temperature sensor has an internal library for reading values. Soil moisture sensor is using the default ESP32 analog signal read method.

```
int readSoilMoisturer(int port) {
  int sensorValue = analogRead(port);
  int moisturePercentage = (100 - ((sensorValue / 4095.00) * 100));
  return moisturePercentage;
}
```

Figure 4.7: Soil Moisture Value Reading Method

```
float readTempreture(int port) {
  float t = dht.readTemperature();
  return t;
}

float readHumidity(int port) {
  float t = dht.readHumidity();
  return t;
}
```

Figure 4.8:Tempreture and Humidity Value Reading Method

The NPK sensor responds exclusively to RS485 Modbus commands in order to provide values for each level of nitrogen, phosphorus, and potassium. The NPK sensor responds to the inquiry frame with 8 hexadecimal values that include the values for nitrogen (N), phosphorus (P), and potassium (K). This communication operates through serial communication using an RS486 to TTL converter.

Request frames for N, P, K values as below

```
const byte nitro[] = { 0x01, 0x03, 0x00, 0x04, 0x00, 0x01, 0xc5, 0xcb };
const byte phos[] = { 0x01, 0x03, 0x00, 0x05, 0x00, 0x01, 0x94, 0x0b };
const byte pota[] = { 0x01, 0x03, 0x00, 0x06, 0x00, 0x01, 0x64, 0x0b };
```

Nitrogen(), phosphorous() and potassium() methods request above inquiry frame to NPK sensor via serial communication. Once response receive it will return back from the method.

```
byte nitrogen() {
  delay(10);
  if (SerialPort.write(nitro, sizeof(nitro)) == 8) {
    for (byte i = 0; i < 7; i++) {
      values[i] = SerialPort.read();
    }
  }
  return values[4];
}
byte phosphorous() {
  delay(10);
  if (SerialPort.write(phos, sizeof(phos)) == 8) {
    for (byte i = 0; i < 7; i++) {
      values[i] = SerialPort.read();
    }
  }
  return values[4];
}
byte potassium() {
  delay(10);
  if (SerialPort.write(pota, sizeof(pota)) == 8) {
    for (byte i = 0; i < 7; i++) {
      values[i] = SerialPort.read();
    }
    Serial.println();
  }
  return values[4];
}
```

Figure 4.9:NPK Values Reading Methods

Publishing sensor values to MQTT topic

Once the values obtained from the sensors, they are transmitted to the backend service using MQTT topic using the following methods.

```
void publishSensorData(int port, float value, String sensorType) {
  // Create a JSON-formatted message
  String jsonMessage = "{\"port\":" + String(port) + ",\"value\":" + String(value) + ",\"sensor\":\"" + sensorType + "\"}";

  // Convert String to char array for publishing
  char messageBuffer[jsonMessage.length() + 1];
  jsonMessage.toCharArray(messageBuffer, sizeof(messageBuffer));

  // Publish the message to the specified topic
  client.publish("SWFS/sensor-data", messageBuffer);
}

void publishNPKSensorData(int port, String sensorType, int N, int P, int K) {
  // Create a JSON-formatted message
  String jsonMessage = "{\"port\":" + String(port) + ",\"n\":" + String(N) + ",\"p\":" + String(P) + ",\"k\":" + String(K) + ",\"sensor\":\"" + sensorType + "\"}";

  // Convert String to char array for publishing
  char messageBuffer[jsonMessage.length() + 1];
  jsonMessage.toCharArray(messageBuffer, sizeof(messageBuffer));

  // Publish the message to the specified topic
  client.publish("SWFS/sensor-data", messageBuffer);
}
```

Figure 4.10: Sensor Values Publishing Method

4.3. Software Implementation

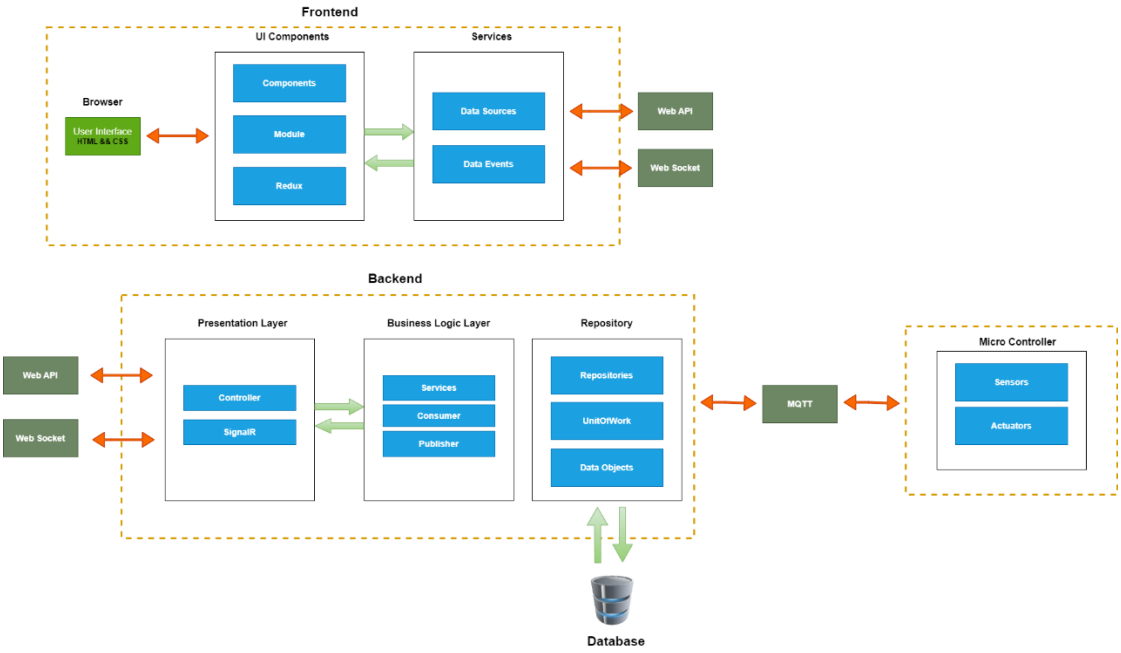


Figure 4.11: Front-end and Back-end Architecture

System has developed based on Client -Server software architecture. Angular frontend as client and NET backend as service working with web API and web socket communication. Postgres database has used to save all data. MQTT protocol has used to communicate from backend to microcontroller. Both frond end and backend developed focus on maintainability and extensibility for the industry standards.

4.3.1. Frontend Architecture

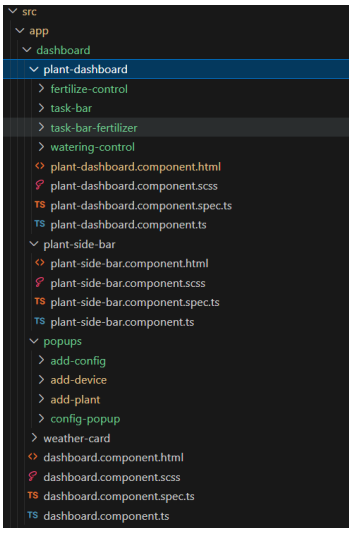


Figure 4.12: Front-end Code Structure

Frontend of the System has developed based on component-based architecture to organize angular application into hierarchy of interconnected components. Each component contains its own view, styles, logic and metadata that working together to render unit of application UI. All these units have bundle together to produce functional application.

Services and Dependency Injections

All http calls have coded inside service classes and injected those classes to component with dependency injections. Dependency injection allow to share class instances between components to consume its methods making system more modular and efficient.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { environment } from 'src/environments/environment';

You, last month | 2 authors (Rishmi Shehan and others)
@Injectable({
  providedIn: 'root'
})
export class PlantService {       Rishmi Shehan, 5 months ago • added main ui c
  private apiUrl = environment.apiUrl;
  constructor(private http: HttpClient) { }

  savePlant(plantData: any): Observable<any> {
    return this.http.post<any>(`${this.apiUrl}/plant`, plantData);
  }
  getPlants(): Observable<any> {
    return this.http.get<any>(`${this.apiUrl}/all-plants`);
  }
  getPlant(plantId:any): Observable<any> {
    return this.http.get<any>(`${this.apiUrl}/plant?plantid=${ plantId }`);
  }
  updatePlant(plantData: any): Observable<any> {
    return this.http.post<any>(`${this.apiUrl}/update`, plantData);
  }
}
```

Figure 4.13: Service Class

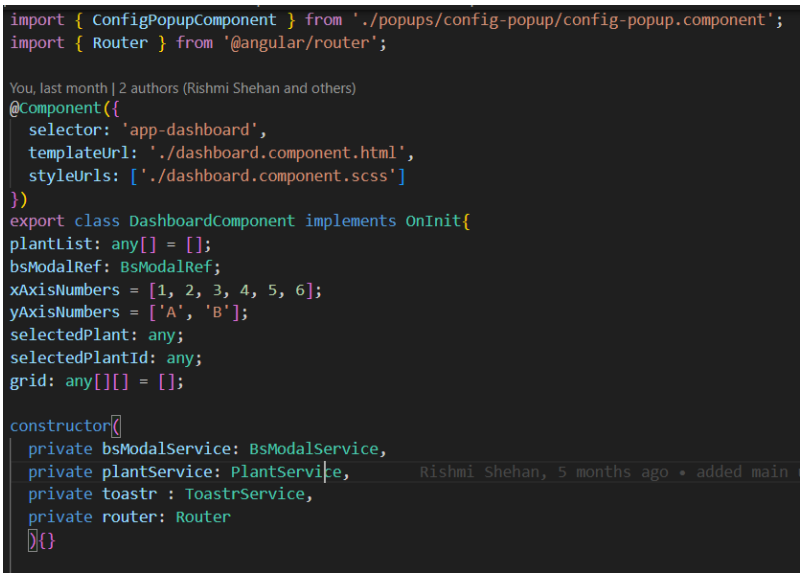


Figure 4.14: Constructor Dependancy Injection

As per the Figure 4.13 and Figure 4.14 http API request has maintained inside plant service and it has injected to the required component constructor. Component can receive and send data with the use of RXJS subscription as Figure 4.15.

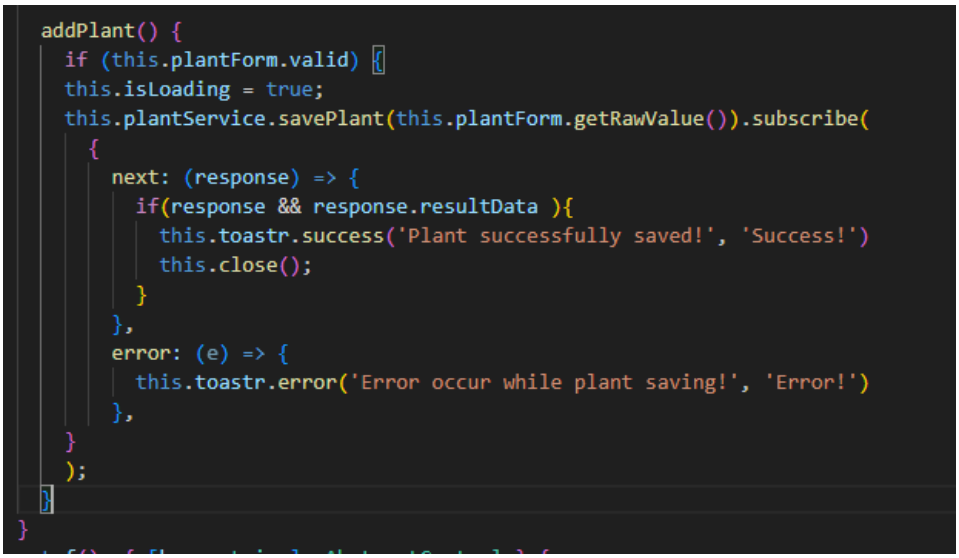


Figure 4.15: Add Plant Method

Redux State Management

Redux provide capability to hold entire state of the application with single store. Redux store has implemented in application to keep real-time data in centralized location. Application obtains real-time data such as sensor values from store with the use RXJS subscriptions. Application update component data at every time state has change.

SignalR

SignalR is a library from Microsoft to helps to minimize complexity of adding real-time web functionality to Application. SingalR provide NPM package to make compatible angular applications to handle real-time communication which happens on the top of web sockets. It is a two-way communication between server and its connected clients that server can push content to its client instantly.

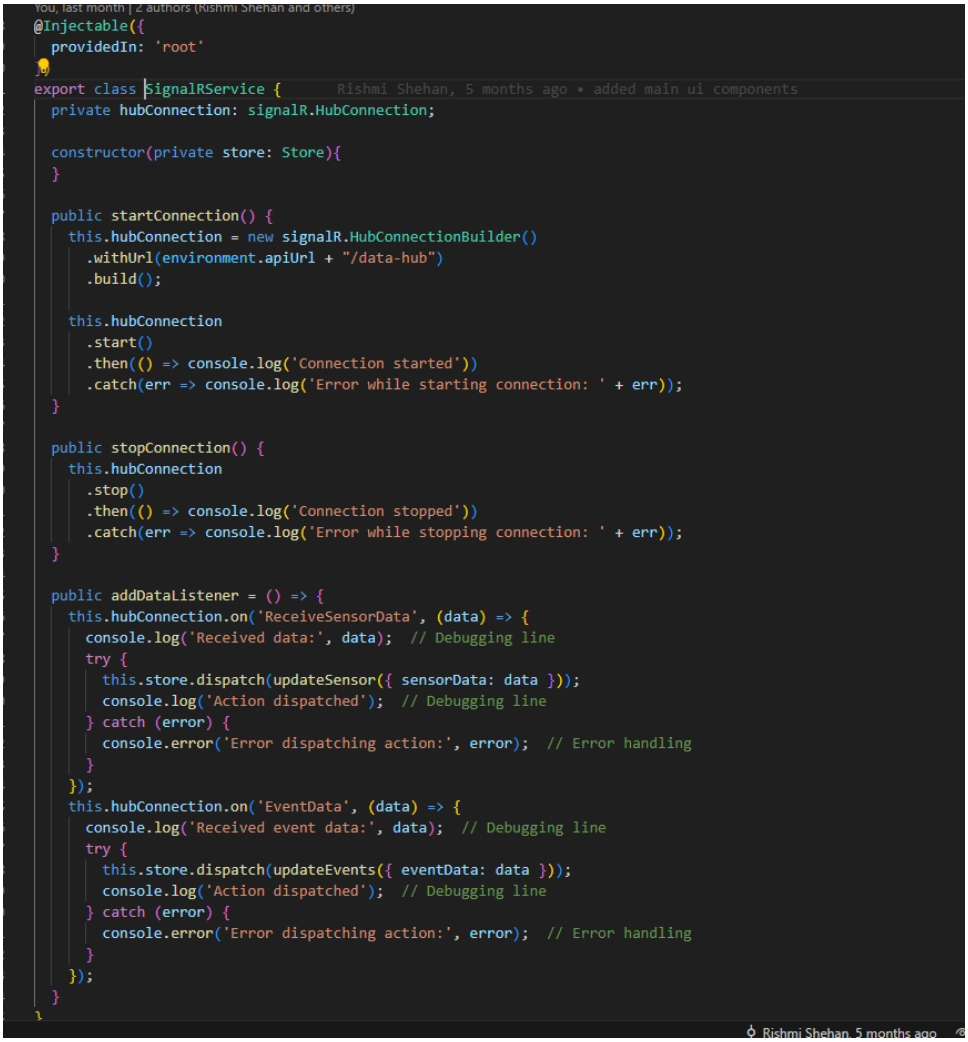
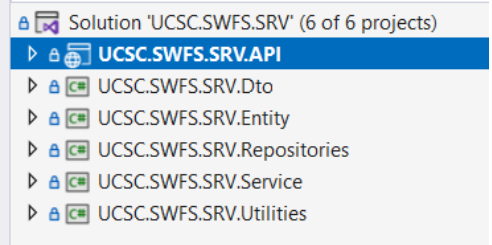


Figure 4.16: Redux State Management

As per the Figure 4.16, Application will listen to the web socket and it will dispatch action to redux store whenever data hub receives a data object. Once get notified from listener store will update its state to different latest state. For every change of state angular component get updated with the latest data across the application.

4.3.2. Backend Architecture

NET core backend implemented based on repository pattern. Classes have organized into layered architecture for better maintainability. NET core application contains three main layers API, services, repositories and three minor layers to hold entities, data transfer objects, utilities.



API Layer

Contains all web API methods and working as presentation layer for request and respond by communicated directly with the business logic layer. HTTP controller methods and service layer interaction describe by following figure. Services have injected to the controller using dependency injections.

```
namespace UCSC.SWFS.SRV.API.Controllers
{
    1 reference | Rishmi Shehan, 39 days ago | 1 author, 2 changes
    public class PlantController : ControllerBase
    {
        private readonly IPlantService _plantService;
        0 references | Rishmi Shehan, 144 days ago | 1 author, 1 change
        public PlantController(IPlantService plantService)
        {
            _plantService = plantService;
        }

        [HttpGet("all-plants")]
        0 references | Rishmi Shehan, 144 days ago | 1 author, 1 change
        public async Task<ResponseDto<List<PlantDto>>> GetAllPlants()
        {
            return await _plantService.GetAllPlants();
        }
        [HttpGet("plant")]
        0 references | Rishmi Shehan, 39 days ago | 1 author, 1 change
        public async Task<ResponseDto<PlantDto>> SavePlant(int plantId)
        {
            return await _plantService.GetPlant(plantId);
        }

        [HttpPost("plant")]
        0 references | Rishmi Shehan, 39 days ago | 1 author, 2 changes
        public async Task<ResponseDto<PlantDto>> SavePlant([FromBody] PlantDto plant)
        {
            return await _plantService.SavePlant(plant);
        }
        [HttpPost("update")]
        0 references | Rishmi Shehan, 39 days ago | 1 author, 1 change
        public async Task<ResponseDto<PlantDto>> UpdatePlant([FromBody] PlantDto plant)
        {
            return await _plantService.UpdatePlant(plant);
        }

    }
}
```

Figure 4.17: Web API Methods

Service Layer (Business logic layer)

Contain core functionality of the application including logical decisions, data processing, background tasks, real-time communications. This layer act as intermediary between presentation layer and repository layer. Following figure contains Plant service which hold all data fetching and storing methods related to the plant. Service layer communicate to the repository layer that has only access to the data source. Multiple repositories can inject to the service class with the use of constructor dependency injection in order to use data methods in repositories.

```
2 references | Rishmi Shehan, 39 days ago | 1 author, 2 changes
public class PlantService : IPlantService
{
    private readonly IPlantRepository _plantRepository;
    private readonly IUnitofWork _unitofWork;
    private readonly IMapper _mapper;
    0 references | Rishmi Shehan, 144 days ago | 1 author, 1 change
    public PlantService(IPlantRepository plantRepository, IMapper mapper, IUnitofWork unitofWork)
    {
        _plantRepository = plantRepository;
        _unitofWork = unitofWork;
        _mapper = mapper;
    }

    2 references | Rishmi Shehan, 39 days ago | 1 author, 2 changes
    public async Task<ResponseDto<List<PlantDto>>> GetAllPlants()
    {
        ResponseDto<List<PlantDto>> plantResponseDto = new ResponseDto<List<PlantDto>>();
        plantResponseDto.ErrorList = new List<ErrorInfoDto>();
        try
        {
            List<Plant> plants = await _plantRepository.GetAll().Include(x => x.SensorData).ToListAsync();
            if (plants != null && plants.Count > 0)
            {
                var plantDtoList = _mapper.Map<List<PlantDto>>(plants);
                plantResponseDto.ResultData = plantDtoList;
            }
            else
            {
                plantResponseDto.ResultData = null;
            }
        }
        catch (Exception ex)
        {
            plantResponseDto.ErrorList.Add(new ErrorInfoDto
            {
                ErrorCode = "SMFS_ERR_02",
                ErrorMessage = UserResource.SMFS_ERR_02
            });
            throw ex;
        }
```

Figure 4.18: Service Layer Method

Apart from data handling, this layer responsible for real-time communication, MQTT communication and background tasks.

Repository Layer

Data access layer on the other hand repository layer provide direct access to the database and contain logics and method for inserting and updating data. Each database table contain its own repository. Each repository can only access its relevant data table. Repository classes coded with generic way with the using interfaces, entity framework and unit of work. Unit of work handle transaction and make sure all changes to the entities in the domain model are applied together in a single transaction to maintain data integrity and consistency. Following figure shows generic repository method written with the use of entity framework and Unit of work implementation.

```
public class Repository<TEntity> : IRepository<TEntity> where TEntity : class
{
    protected readonly DbContext _dbContext;
    protected readonly DbSet<TEntity> _dbSet;
    private readonly IRequestHeader _requestHeader;

    1 reference | Rishmi Shehan, 265 days ago | 1 author, 1 change
    public Repository(DbContext dbContext, IRequestHeader requestHeader)
    {
        _dbContext = dbContext;
        _dbSet = dbContext.Set<TEntity>();
        _requestHeader = requestHeader;
    }
    1 reference | Rishmi Shehan, 265 days ago | 1 author, 1 change
    public async Task<IQueryable<TEntity>> GetAllAsync(Expression<Func<TEntity, bool>> predicate = null)
    {
        IQueryable<TEntity> query = await Task.FromResult(_dbSet.AsQueryable());
        if (predicate != null)
        {
            query = query.Where(predicate);
        }
        return query;
    }

    2 references | Rishmi Shehan, 265 days ago | 1 author, 1 change
    public IQueryable<TEntity> GetAll(Expression<Func<TEntity, bool>> predicate = null)
    {
        IQueryable<TEntity> query = _dbSet;
        if (predicate != null)
        {
            query = query.Where(predicate);
        }
        return query;
    }

    2 references | Rishmi Shehan, 265 days ago | 1 author, 1 change
    public async Task<TEntity> InsertAsync(TEntity entity)
    {
        Helper.SetBasePropertiesOnInsert(entity, _requestHeader);
        await _dbSet.AddAsync(entity);
        return entity;
```

Figure 4.19:Repository Methods

Background Tasks

Scheduler task has defined for watering and fertilization process. At each 30 seconds it will check for plant status and do watering and fertilization as per the requirement.

```
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{

    // Your background logic goes here
    while (!stoppingToken.IsCancellationRequested)
    {
        using (var scope = _serviceScopeFactory.CreateScope())
        {

            IScheduleService _scheduleService = scope.ServiceProvider.GetRequiredService<IScheduleService>();
            // Perform the tasks or operations you need
            Console.WriteLine($"Background service is running @ {@DateTime.UtcNow}");

            await _scheduleService.WateringPlantsSchedulerAsync();
            // Delay for a certain duration before running the next iteration
            await Task.Delay(TimeSpan.FromSeconds(30), stoppingToken);

            await _scheduleService.FertilizingPlantSchedulerAsync();
            // Delay for a certain duration before running the next iteration
            await Task.Delay(TimeSpan.FromSeconds(30), stoppingToken);
        }

    }
}
```

Figure 4.20: Background Tasks

As per the Figure 4.20, two methods handling critical functionality in the system. It will check for plant water and fertilizer level from the sensors and then perform watering and fertilization process by communicating to microcontroller. At the same time all events are notify to clients via SignalR to show progress to frontend application users real-time.

MQTT Integration

Consumer and publisher have implemented to handle data publish and consume with microcontroller. At each 60 seconds microcontroller publish all sensor data to "`SWFS-sensor-data" and` service will consume data object and process. On the other hand, backend service will publish message to MQTT topic "`SWFS -WateringTasks", "SWFS-FertilizingTasks"` whenever plant required watering or fertilization.

```
0 references | Rishmi Shehan, 145 days ago | 1 author, 1 change
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
    var options = new MqttClientOptionsBuilder()
        .WithTcpServer("test.mosquitto.org", 1883)
        .Build();

   _mqttClient.ApplicationMessageReceivedAsync +=  HandleReceivedApplicationMessage;

    await _mqttClient.ConnectAsync(options, stoppingToken);

    var subscribeOptions = new MqttClientSubscribeOptionsBuilder()
        .WithTopicFilter("SWFS-sensor-data")
        .Build();

    await _mqttClient.SubscribeAsync(subscribeOptions, stoppingToken);
}
```

Figure 4.21: Subscribe MQTT Topic

```
            TaskDto wateringTask = new TaskDto()
            {
                PlantId = plant.PlantId,
                TaskType = "WATERING",
                Port   = waterPump.Port,
                Duration = wateringDuration
            };
            performedTime = DateTime.UtcNow;
            await _MQTTBrockerService.PublishMessageToMqttBroker("WateringTasks", JsonConvert.SerializeObject(wateringTask));
            EventDataDto wateringEventDataDto = new EventDataDto()
            {
                Id = Guid.NewGuid(),
                Type = "WATTERING",
                PlantID = plant.PlantId,
                EventType = "AUTOMATED_WATERING",
                StartAt = performedTime,
                Duration = wateringDuration
            };
            await _dataHubContext.Clients.All.SendAsync("EventData", eventDataDto);

            if (plant.WateringScheduleType == nameof(WateringScheduleType.ROUTING))
            {
                plant.NextWateringTime = NextWateringDate(plant);
            }
            plant.LastWatered = performedTime.HasValue ? performedTime.Value : null;

            _plantRepository.Update(plant);
            await _unitofWork.SaveChangesAsync();
        }
    }
```

Figure 4.22:Publish message to MQTT topic

4.3.3. Machine Learning Integration

Few data models have been trained using the dataset from an external source. Due to the lack of historical data in the system, an external dataset was utilized to train the models.

	A	B	C	D	E	F
1	CropType	CropDays	SoilMoistur	Temperatı	Humidity	Irrigation
2	Wheat	10	400	30	15	0
3	Wheat	7	200	30	32	0
4	Wheat	9	300	21	28	0
5	Wheat	3	500	40	22	0
6	Wheat	2	700	23	34	0
7	Wheat	6	800	21	29	0
8	Wheat	5	500	33	26	0
9	Wheat	8	350	21	28	0
10	Wheat	11	123	17	45	0
11	Wheat	12	543	25	53	0
12	Wheat	13	425	33	22	0

Figure 4.23 Watering Dataset

The dataset shown in Figure 4.23 was utilized for training the model. It was divided into two separate datasets for training and evaluation purposes. Figure 4.24 illustrates the algorithm utilized for training the watering model and its output, which is the accuracy of the data model after evaluation.

```
public ModelEvaluationMetrics TrainAndSaveModel()
{
    try
    {
        IDataView dataView = _mlContext.Data.LoadFromTextFile<CropWateringData>
            (Constants.basePath + Constants.WateringDataSet, hasHeader: true, separatorChar: ',');
        DataOperationsCatalog.TrainTestData dataSplit = _mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);
        IDataView trainingData = dataSplit.TrainSet;
        IDataView testData = dataSplit.TestSet;

        var dataProcessPipeline = _mlContext.Transforms.Concatenate("Features", nameof(CropWateringData.CropDays),
            nameof(CropWateringData.SoilMoisture), nameof(CropWateringData.Temperature), nameof(CropWateringData.Humidity))
            .Append(_mlContext.BinaryClassification.Trainers.SdcaLogisticRegression(nameof(CropWateringData.Irrigation), featureColumnName: "Features"));

        var model = dataProcessPipeline.Fit(trainingData);

        var predictions = model.Transform(testData);
        var metrics = _mlContext.BinaryClassification.Evaluate(predictions, labelColumnName: nameof(CropWateringData.Irrigation));

        _mlContext.Model.Save(model, trainingData.Schema, Constants.basePath + Constants.WateringModelSavePath);

        // Create and return the metrics object
        return new ModelEvaluationMetrics
        {
            Accuracy = metrics.Accuracy,
            AreaUnderRocCurve = metrics.AreaUnderRocCurve,
            F1Score = metrics.F1Score
        };
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred: {ex.Message}");
        throw ex;
    }
}
}
```

Figure 4.24 Model Training Algorithm

Once the training completed, the trained model is saved into a file and it will be retrieved whenever the system needs to make data predictions, as shown in figure 4.25.

```
public WateringPrediction Predict(CropWateringData input)
{
    DataViewSchema modelSchema;
    _model = _mlContext.Model.Load(Constants.basePath + Constants.WateringModelSavePath, out modelSchema);
    var predictionEngine = _mlContext.Model.CreatePredictionEngine<CropWateringData, WateringPrediction>(_model);
    return predictionEngine.Predict(input);
}
```

Figure 4.25 Prediction Function

Based on the return predicted value system determine plant require water or not.

4.4. System Deployment

Entire system deployed in Kubernetes cluster inside Microsoft Azure (AKS). Azure CI/CD pipelines integrated to automate deployment.

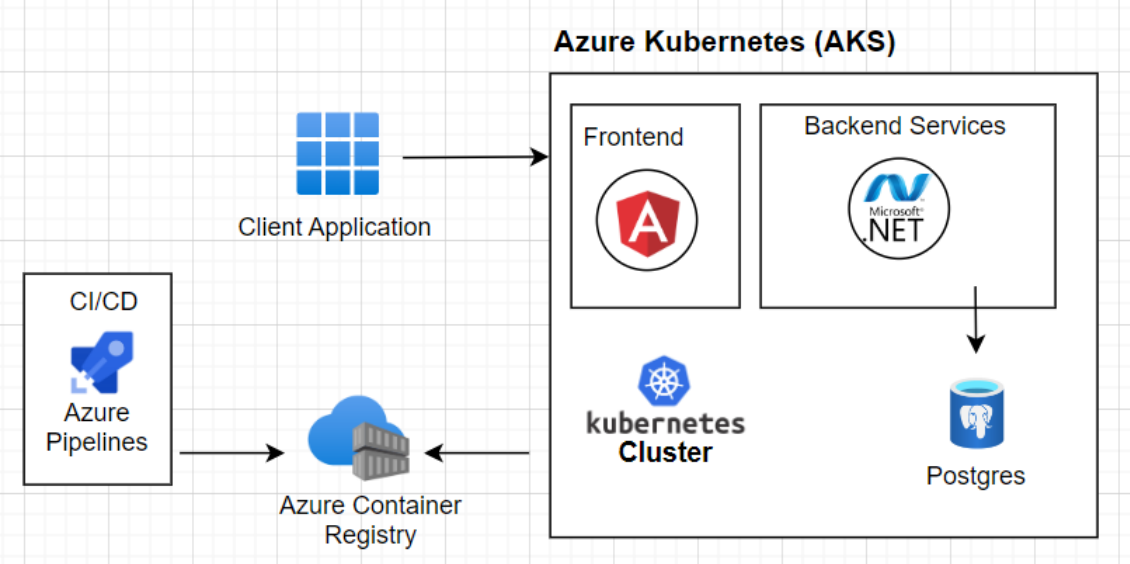


Figure 4.26: Deployment Architecture

Standard_A2_v2 Kubernetes cluster with 4GB CPU and 2GB RAM used to deploy frontend and backend services. Azure pipeline build versions on top of Linux base image and then built docker images will push to Azure container registry. Finaly images will be taken from container registry and deploy in Kubernetes cluster. Postgres service also deploy inside Kubernetes cluster.

5 CHAPTER - TESTING AND EVALUATION

This section provides an overview of the objectives of the testing process, the main stages it includes, the techniques that can be used for carrying out tests, and the implementation of test plans. Afterwards, it focuses on the test cases, providing comprehensive information on each test's essential requirements and objectives, the required inputs and actions, and the expected results. Additionally, the chapter offers a system evaluation of the system by implementing it in a small-scale pilot project.

5.1. Testing Strategies

A test strategy outlines the process and approach for executing tests on a project, including aspects of both verification and validation and following test strategies used to validate the developed system.

5.1.1. Unit Testing

Unit testing involves thoroughly reviewing and verifying individual components or modules of a web application to ensure their proper functioning in isolation. Unit testing was done in the first phase of software development, and it helped to detect and fix bugs and also improved the robustness and reliability of the system. Developers can ensure the accuracy of individual functions, methods, or classes by concentrating on the most testable components of an application.

5.1.2. Integration Testing

Integration testing is a crucial stage in the software development process that assesses the interaction between hardware and software components of a system. This testing process extends unit testing by evaluating the functioning of various modules or systems when they are integrated. Integration testing involves initiating events from the system, assessing the hardware's response to each event, and assessing the outcome of an event initiated by hardware.

5.1.3. System Testing

Validate that the system meets the initial requirement and behaves according to expected outcomes. This process covers both functional and non-functional requirements.

5.2. Test Cases

Table 5.1 contains some of the main test cases related to the UI components.

Test #	Test Description	Input/Action	Expected Outcome
1	Add Plant with all valid inputs	Plant Name: "Rose", Plant Type: "Flower", Planting Date: "2023-11-01", Water Level: "25", Fertilizer Level: "20", Recommended Temperature Min: "28", Max: "30", Soil Moisture: "40", N: "20", P: "20", K: "20"	Plant is added successfully, and a success message is displayed. New plant added to main grid
2	Add Plant without Plant Name	Plant Name: "", remaining fields valid	Validation error message for "Plant Name is required." is displayed.
3	Save button disabled when form is invalid	Leaves required fields empty and tries to click "Save"	The "Save" button is disabled until all required fields are valid.
4	Add Device with valid data	Device Name: "Temp Sensor 1", Device Type: "TEMPERATURE_SENSOR", Port: 28	Device is added successfully, and a success message is displayed.
5	Add Device without Device Name	Device Name: "", Device Type: "HUMIDITY_SENSOR", Port: 2	Validation error message for "Device Name is required." is displayed.

6	Save button disabled when form is invalid	Leaves required fields empty and tries to click "Save"	The "Save" button is disabled until all required fields are valid.
7	Map Device to Plant	Search for a plant by ID or Name, select it, and click "Map" button	The selected plant is mapped to the device, and its details are displayed under "Selected Plant."
9	Add Global Configuration with valid data	Configuration Type: "Global Configuration", Configuration Name: "Max Temperature", Value Type: "number", Value: "30"	Configuration is added successfully, and a success message is displayed.
10	Add Global Configuration without a name	Configuration Type: "Plant Configuration", Name: ""	Validation error message for "Configuration Name is required." is displayed.
11	Add Plant Configuration with valid data	Configuration Type: "Plant Configuration", Configuration Name: "Max Temperature", Value Type: "number", Value: "30"	Configuration is added successfully, and a success message is displayed.
12	Add Device Configuration with valid data	Configuration Type: "Device Configuration",Configuration Name: "Max Temperature", Value Type: "number", Value: "30"	Configuration is added successfully, and a success message is displayed.
13	Added plant configuration reflect in config grid	None	Configuration appear in config grid
14	Added device configuration reflect in config grid	None	Configuration appear in config grid

15	Delete Configuration	Click respective row delete button	Configuration deleted from grid
16	Display plants in the grid	Load the grid view	Each cell in the grid should correctly display a plant card if a plant exists, or an empty card otherwise.
17	Select a plant from the grid	Click on a plant card	The clicked plant card becomes highlighted (selected), indicating it has been selected.
18	Add a new plant from an empty grid cell	Click on an empty grid cell	Triggers the add plant functionality, potentially opening a form to input new plant details.
19	Display correct plant details	Hover or click on a plant card	The plant's details (e.g., plant name) are correctly displayed on the card.
20	Validate plant selection functionality	Select different plants sequentially	Each selection clears the previous selection and highlights the new selection.
21	Validate dynamic grid updates	Add or remove plants and observe grid updates	The grid should dynamically update to reflect changes, showing or hiding plant cards and empty cells.
22	Display Sensor Readings	Load the control section	Sensor readings for temperature, humidity, and moisture are displayed correctly with appropriate icons and values.

23	Update Sensor Readings	Trigger an update (e.g., change sensor locations)	Updated sensor readings are fetched and displayed real-time accurately.
25	Validate Temperature Reading Display	Check the temperature reading	The temperature is displayed correctly with the temperature icon and should be in °C.
26	Validate Humidity Reading Display	Check the humidity reading	The humidity is displayed correctly with the humidity icon and should be represented as a percentage (%).
27	Validate Moisture Reading Display	Check the moisture reading	The moisture is displayed correctly with the moisture icon and should be represented as a percentage (%).
28	Toggle Watering Mode	Click on "Automated" and then "Manual" radio buttons under the "Actions" section.	The interface updates to show options specific to each selected watering mode.
29	Save Automated Watering Mode	Select "Automated" mode and trigger an interface action to save/update the plant configuration.	The system saves the "Automated" watering mode as the setting for the plant and maintains this setting upon page reload or navigation.
30	Save Manual Watering Mode	Select "Manual" mode and trigger an interface action to save/update the plant configuration.	The system saves the "Manual" watering mode as the setting for the plant and maintains this setting upon page reload or navigation.

31	Select Automated Watering Schedule Type	In "Automated" mode, select "Sensor Based" and "Routing" options for watering schedule type.	Options relevant to the selected schedule type are displayed appropriately.
32	Update Watering Schedule in Automated Mode	Choose a schedule from the dropdown when "Routing" is selected as the schedule type and trigger an update.	The chosen watering schedule is saved and will be applied to the plant's watering routine.
33	Select Water Level Type in Automated Mode	Select between "Sensor Based" and "Manual" water level types in "Automated" mode and trigger an update.	The interface updates to display input fields appropriate for the selected water level type.
34	Input Manual Water Level in Automated Mode	Enter a value in "Manual Water Level (ml)" under "Automated" watering and trigger an update.	The manual water level input is saved for the plant's watering configuration.
35	Toggle Water Level Type in Manual Mode	Click between "Sensor Based" and "Manual" under "Manual" watering mode.	The interface updates to reflect the selected water level type, showing sensor data or allowing manual input.
36	Refresh Sensor-Based Water Level in Manual Mode	Click the "Refresh" button under "Sensor Based" water level type in "Manual" mode.	The displayed sensor-based water level is updated to the current reading.
37	Input Manual Water Level in Manual Mode	Enter a value in "Manual Water Level (ml)" under "Manual" watering mode and trigger an update.	The manual water level input is saved for the plant's watering configuration.
38	Execute Watering Action	Click the "Start" button under either watering mode.	The system initiates the watering action based on

			the current settings and configuration.
39	Display Sensor Readings for Nitrogen	Navigate to the sensor readings section.	Nitrogen sensor readings are displayed correctly, showing "30mg" as per the latest sensor data.
40	Display Sensor Readings for Phosphorus	Navigate to the sensor readings section.	Phosphorus sensor readings are displayed correctly, showing "40mg" as per the latest sensor data.
41	Display Sensor Readings for Potassium	Navigate to the sensor readings section.	Potassium sensor readings are displayed correctly, showing "155mg" as per the latest sensor data.
42	Display Fertilizing Control Label	Check the top of the fertilizing control div.	The label "Fertilizing Control" is displayed prominently at the top of the control section.
43	Toggle Automated Fertilizing Mode	Select the "Automated" radio button under Actions.	The automated fertilizing options become available for configuration.
44	Toggle Manual Fertilizing Mode	Select the "Manual" radio button under Actions.	The manual fertilizing options become available for configuration.
45	Automated Mode: Select Fertilizing Schedule Type - Sensor Based	In Automated mode, select "Sensor Based" for the Fertilizing Schedule Type.	The system should not display additional options for schedule customization, relying on sensor data.

46	Automated Mode: Select Fertilizing Schedule Type - Routing	In Automated mode, select "Routing" for the Fertilizing Schedule Type.	The system displays options to select fertilizing schedule frequency.
47	Automated Mode: Select Fertilizing Level Type - Manual	In Automated mode, select "Manual" for the Fertilizer Level Type.	Input fields for N, P, and K fertilizer levels become available for manual input.
48	Manual Mode: Verify Fertilizer Level Type Selection	In Manual mode, ensure both "Sensor Based" and "Manual" options are available.	Both options should be selectable with respective adjustments in the UI for entering or displaying fertilizer levels.

Table 5.1: Test Cases for UI components

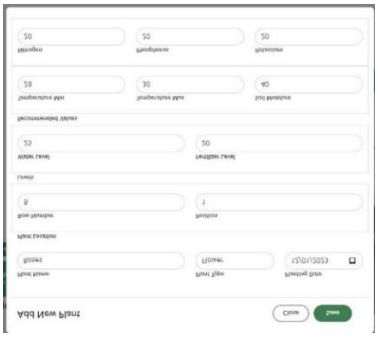
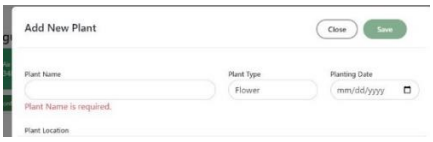
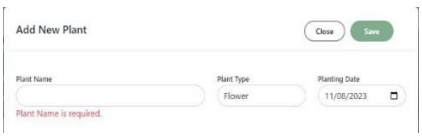
Table 5.2 contains test cases for hardware components.

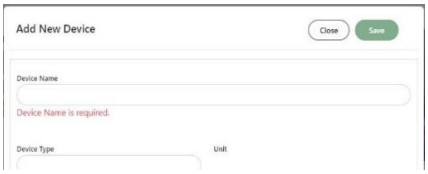
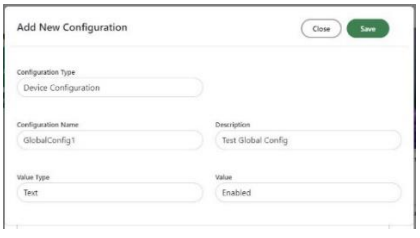
Test #	Test Description	Input/Action	Expected Outcome
49	Water Pump Activation Test	Change watering type to manual and trigger watering task with 50ml water amount	Water pump activates and irrigation begins. System logs the activation event.
50	Fertilization Pumps Accuracy Test	Program specific N, P, K fertilizer release.	Each pump dispenses the exact programmed amount of fertilizer.
51	Temperature Sensor Calibration Test	Expose to controlled temperature.	Sensor readings align with calibrated reference thermometer readings.
52	Humidity Sensor Response Test	Place in known humidity levels.	Sensor readings reflect in system
53	Soil Moisture Sensor Functionality Test	Insert into soil with varying moisture.	Sensor readings reflect in system
54	NPK Sensor test	Expose to various soil types	Sensor readings reflect in system

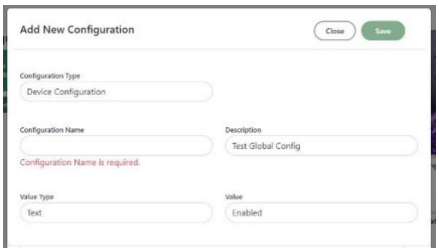
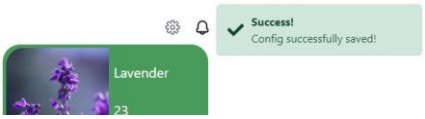
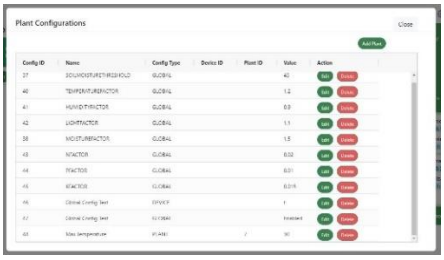
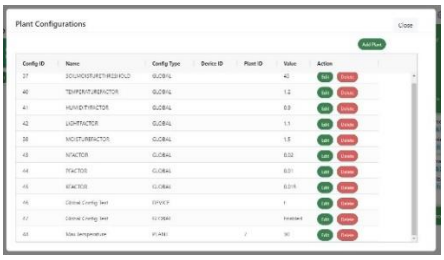
Table 5.2: Test Cases for Hardware Components

5.3. Test Results

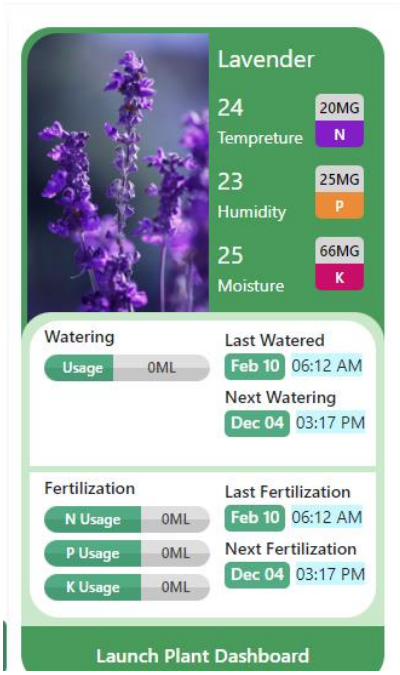
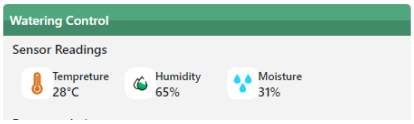
Table 5.3 contains the results for UI test cases.

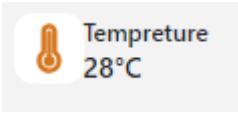
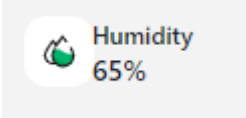
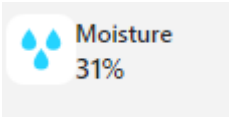
Test #	Test Description	Input/Action	Expected Outcome	Test Result
1	Add Plant with all valid inputs	Plant Name: "Rose", Plant Type: "Flower", Planting Date: "2023-11-01", Water Level: "25", Fertilizer Level: "20", Recommended Temperature Min: "28", Max: "30", Soil Moisture: "40", N: "20", P: "20", K: "20"	Plant is added successfully, and a success message is displayed. New plant added to main grid	  
2	Add Plant without Plant Name	Plant Name: "", remaining fields valid	Validation error message for "Plant Name is required." is displayed.	
3	Save button disabled when form is invalid	Leaves required fields empty and tries to click "Save"	The "Save" button is disabled until	

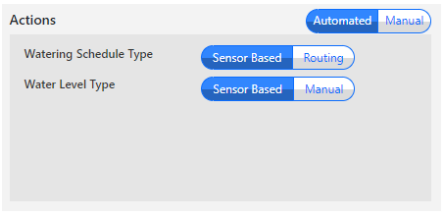
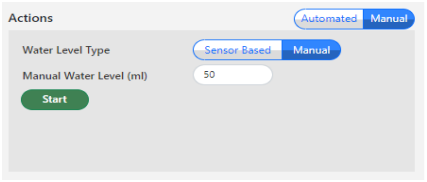
			all required fields are valid.	
4	Add Device with valid data	Device Name: "Temp Sensor 1", Device Type: "TEMPERATURE_SENSOR",Port: 28	Device is added successfully, and a success message is displayed.	
5	Add Device without Device Name	Device Name: "", Device Type: "HUMIDITY_SENSOR" , Port: 2	Validation error message for "Device Name is required." is displayed.	
6	Save button disabled when form is invalid	Leaves required fields empty and tries to click "Save"	The "Save" button is disabled until all required fields are valid.	
7	Map Device to Plant	Search for a plant by ID or Name, select it, and click "Map" button	The selected plant is mapped to the device, and its details are displayed under "Selected Plant."	
9	Add Global Configuration with valid data	Configuration Type: "Global Configuration", Configuration Name: "Max Temperature", Value Type: "number", Value: "30"	Configuration is added successfully, and a success message is displayed.	

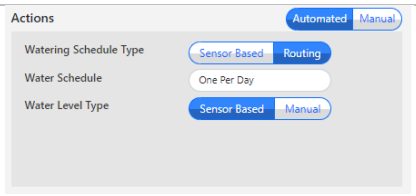
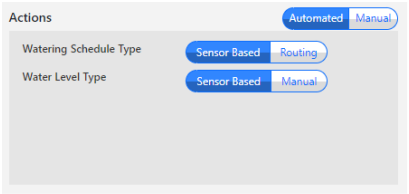
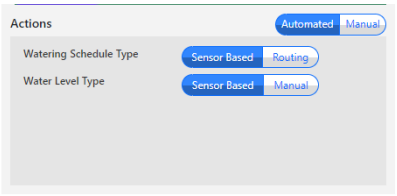
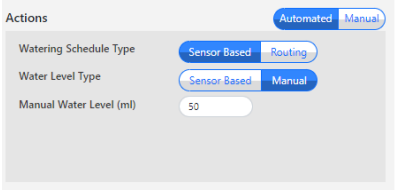
10	Add Global Configuration without a name	Configuration Type: "Plant Configuration", Name: ""	Validation error message for "Configuration Name is required." is displayed	
11	Add Plant Configuration with valid data	Configuration Type: "Plant Configuration", Configuration Name: "Max Temperature", Value Type: "number", Value: "30"	Configuration is added successfully, and a success message is displayed.	
12	Add Device Configuration with valid data	Configuration Type: "Device Configuration",Configuration Name: "Max “ Temperature",Value Type: "number",Value: "30"	Configuration is added successfully, and a success message is displayed.	
13	Added plant configuration reflect in config grid	None	Configuration appear in config grid	
14	Added device configuration reflect in config grid	None	Configuration appear in config grid	

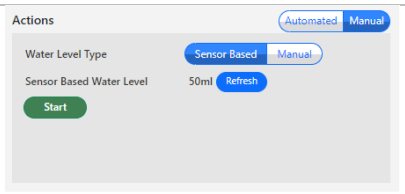
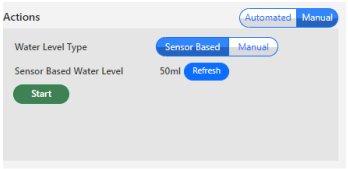
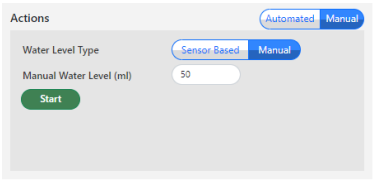
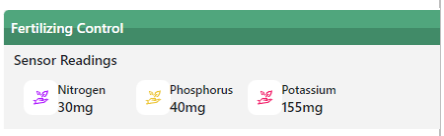
15	Delete Configuration	Click respective row delete button	Configuration deleted from grid	Configuration is successfully removed from the grid.
16	Display plants in the grid	Load the grid view	Each cell in the grid should correctly display a plant card if a plant exists, or an empty card otherwise.	
17	Select a plant from the grid	Click on a plant card	The clicked plant card becomes highlighted (selected), indicating it has been selected.	
18	Add a new plant from an empty grid cell	Click on an empty grid cell	Triggers the add plant functionality, potentially opening a form to input new plant details.	

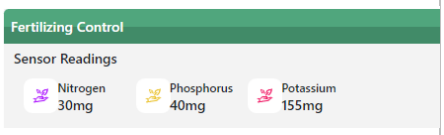
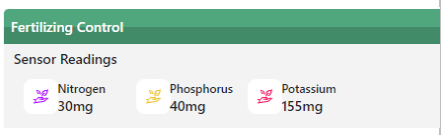
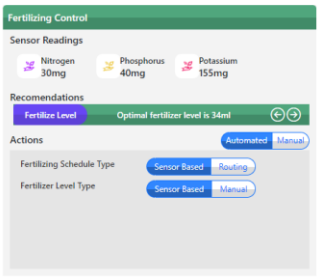
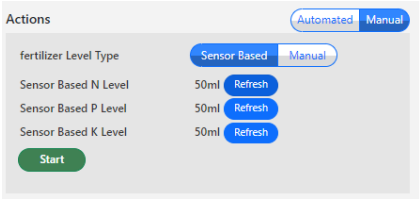
29	Display correct plant details	Hover or click on a plant card	The plant's details (e.g., plant name) are correctly displayed on the card.	
20	Validate plant selection functionality	Select different plants sequentially	Each selection clears the previous selection and highlights the new selection.	Verified Each selection clears the previous selection and highlighted the new section
21	Validate dynamic grid updates	Add or remove plants and observe grid updates	The grid should dynamically update to reflect changes, showing or hiding plant cards and empty cells.	Verified grid update for each event
22	Display Sensor Readings	Load the control section	Sensor readings for temperature, humidity, and moisture are displayed correctly with	

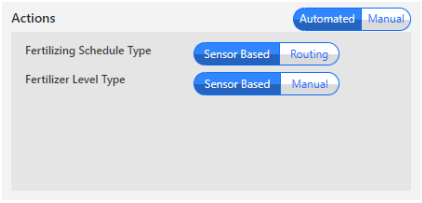
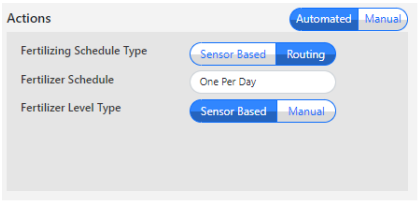
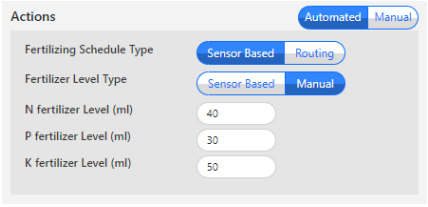
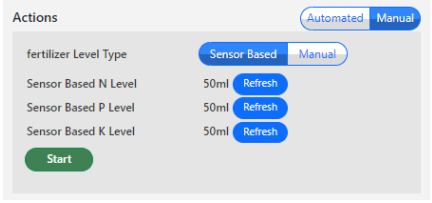
			appropriate icons and values.	
23	Update Sensor Readings	Trigger an update (e.g., change sensor locations)	Updated sensor readings are fetched and displayed real-time accurately.	Confirmed that real-time sensor reading updates are working
25	Validate Temperature Reading Display	Check the temperature reading	The temperature is displayed correctly with the temperature icon and should be in °C.	
26	Validate Humidity Reading Display	Check the humidity reading	The humidity is displayed correctly with the humidity icon and should be represented as a percentage (%).	
27	Validate Moisture Reading Display	Check the moisture reading	The moisture is displayed correctly with the moisture icon and should be represented as a percentage (%).	

28	Toggle Watering Mode	Click on "Automated" and then "Manual" radio buttons under the "Actions" section.	The interface updates to show options specific to each selected watering mode.	 
29	Save Automated Watering Mode	Select "Automated" mode and trigger an interface action to save/update the plant configuration.	The system saves the "Automated" watering mode as the setting for the plant and maintains this setting upon page reload or navigation.	Confirmed that the saved mode is visible again after a page refresh
30	Save Manual Watering Mode	Select "Manual" mode and trigger an interface action to save/update the plant configuration.	The system saves the "Manual" watering mode as the setting for the plant and maintains this setting upon page reload or navigation.	Confirmed that the saved mode is visible again after a page refresh
31	Select Automated Watering Schedule Type	In "Automated" mode, select "Sensor Based" and "Routing" options for watering schedule type.	Options relevant to the selected schedule type	

			are displayed appropriately.	
32	Update Watering Schedule in Automated Mode	Choose a schedule from the dropdown when "Routing" is selected as the schedule type and trigger an update.	The chosen watering schedule is saved and will be applied to the plant's watering routine.	
33	Select Water Level Type in Automated Mode	Select between "Sensor Based" and "Manual" water level types in "Automated" mode and trigger an update.	The interface updates to display input fields appropriate for the selected water level type.	
34	Input Manual Water Level in Automated Mode	Enter a value in "Manual Water Level (ml)" under "Automated" watering and trigger an update.	The manual water level input is saved for the plant's watering configuration.	
35	Toggle Water Level Type in Manual Mode	Click between "Sensor Based" and "Manual" under	The interface updates to reflect the selected water	

		"Manual" watering mode.	level type, showing sensor data or allowing manual input.	
36	Refresh Sensor-Based Water Level in Manual Mode	Click the "Refresh" button under "Sensor Based" water level type in "Manual" mode.	The displayed sensor-based water level is updated to the current reading.	
37	Input Manual Water Level in Manual Mode	Enter a value in "Manual Water Level (ml)" under "Manual" watering mode and trigger an update.	The manual water level input is saved for the plant's watering configuration.	
38	Execute Watering Action	Click the "Start" button under either watering mode.	The system initiates the watering action based on the current settings and configuration.	Confirmed functionality of the watering pump for a certain period of time.
39	Display Sensor Readings for Nitrogen	Navigate to the sensor readings section.	Nitrogen sensor readings are displayed correctly, showing "30mg" as per the latest sensor data.	

40	Display Sensor Readings for Phosphorus	Navigate to the sensor readings section.	Phosphorus sensor readings are displayed correctly, showing "40mg" as per the latest sensor data.	
41	Display Sensor Readings for Potassium	Navigate to the sensor readings section.	Potassium sensor readings are displayed correctly, showing "155mg" as per the latest sensor data.	
42	Toggle Automated Fertilizing Mode	Select the "Automated" radio button under Actions.	The automated fertilizing options become available for configuration.	
43	Toggle Manual Fertilizing Mode	Select the "Manual" radio button under Actions.	The manual fertilizing options become available for configuration.	

44	Automated Mode: Select Fertilizing Schedule Type - Sensor Based	In Automated mode, select "Sensor Based" for the Fertilizing Schedule Type.	The system should not display additional options for schedule customization, relying on sensor data.	
45	Automated Mode: Select Fertilizing Schedule Type - Routing	In Automated mode, select "Routing" for the Fertilizing Schedule Type.	The system displays options to select fertilizing schedule frequency.	
46	Automated Mode: Select Fertilizing Level Type - Manual	In Automated mode, select "Manual" for the Fertilizer Level Type.	Input fields for N, P, and K fertilizer levels become available for manual input.	
47	Manual Mode: Verify Fertilizer Level Type Selection	In Manual mode, ensure both "Sensor Based" and "Manual" options are available.	Both options should be selectable with respective adjustments in the UI for entering or displaying fertilizer levels.	

48	Manual Mode: Input Manual Fertilizer Levels	In Manual mode with Manual level type selected, enter values for N, P, and K levels.	The entered values are updated.	The entered values are reflected after the page is reloaded and verified.

Table 5.3 Test Results for UI component test cases

Table 5.4 contains test results for hardware components-related test cases.

Test #	Test Description	Input/Action	Expected Outcome	Test Result
49	Water Pump Activation Test	Change watering type to manual and trigger watering task with 50ml water amount	Water pump activates and irrigation begins. System logs the activation event.	Confirmed that the relevant relay is activated and the watering pump is functioning.
50	Fertilization Pumps Activation Test	Program specific N, P, K fertilizer release.	Each pump activates and fertilization begins. System logs the activation event.	Confirmed that the relevant relay is activated and that the fertilizer pumps are functioning.
51	Temperature Sensor Response Test		Sensor readings reflect in system	Confirmed system reading temperature values and change value for different temperature levels

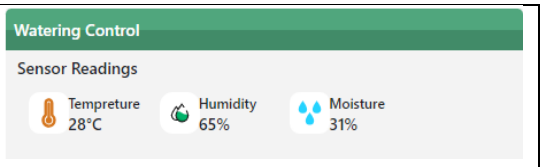
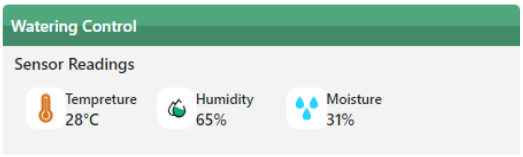
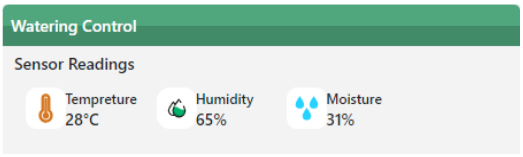
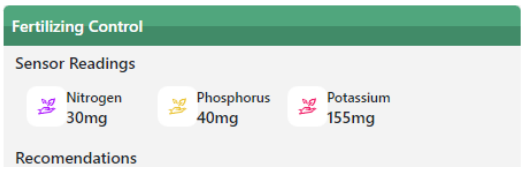
				
52	Humidity Sensor Response Test		Sensor readings reflect in system	Confirmed system reading humidity values and change value for different humidity levels 
53	Soil Moisture Sensor Functionality Test	Insert into soil with varying moisture.	Sensor readings reflect in system	Confirmed system reading moisture values and change value for different soil types 
54	NPK Sensor test	Expose to various soil types	Sensor readings reflect in system	Confirmed system reading NPK values 

Table 5.4:Test Results for Hardware related test cases

5.4. Evaluation

The system was evaluated through a survey and a small-scale pilot project experiment.

5.4.1. Survey

The survey primarily focused on the UI/UX aspect. The system was demonstrated to a select group of experienced individuals in the IT industry and their input was gathered using a Google form. The survey questions are illustrated in figures 5.1, 5.2, and 5.3.

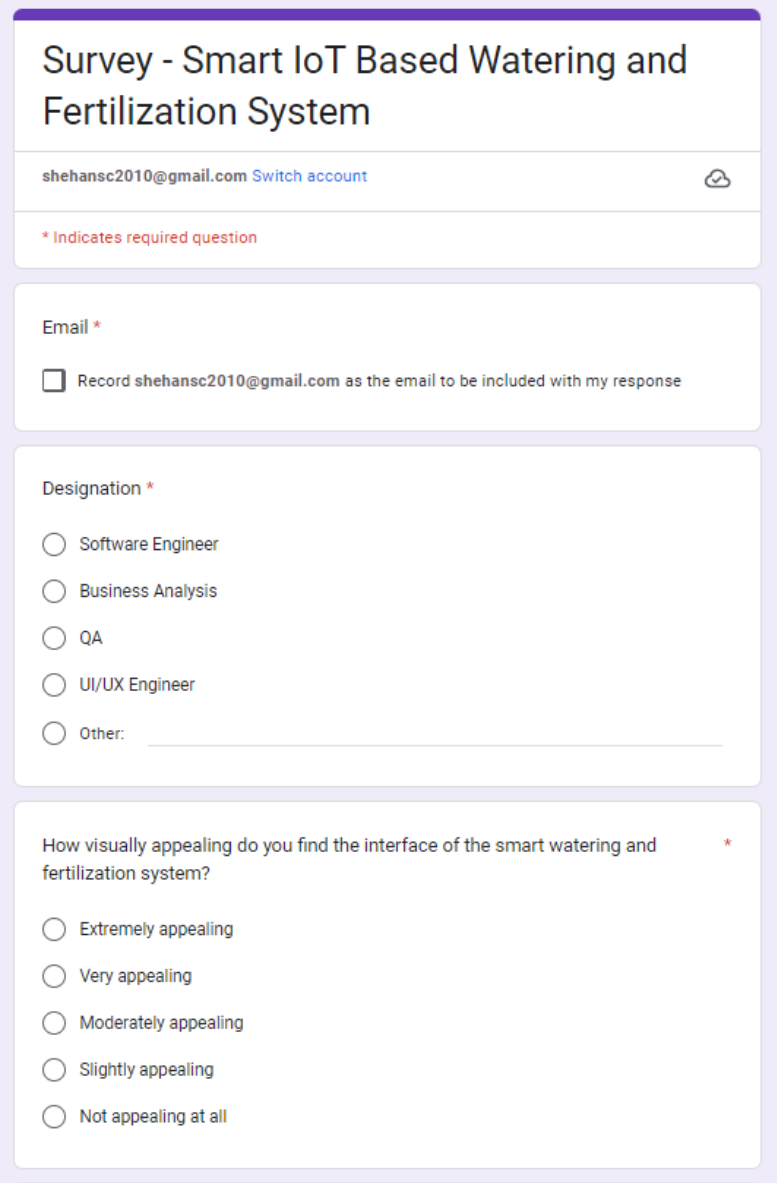


Figure 5.1 Survey - Google Form page1

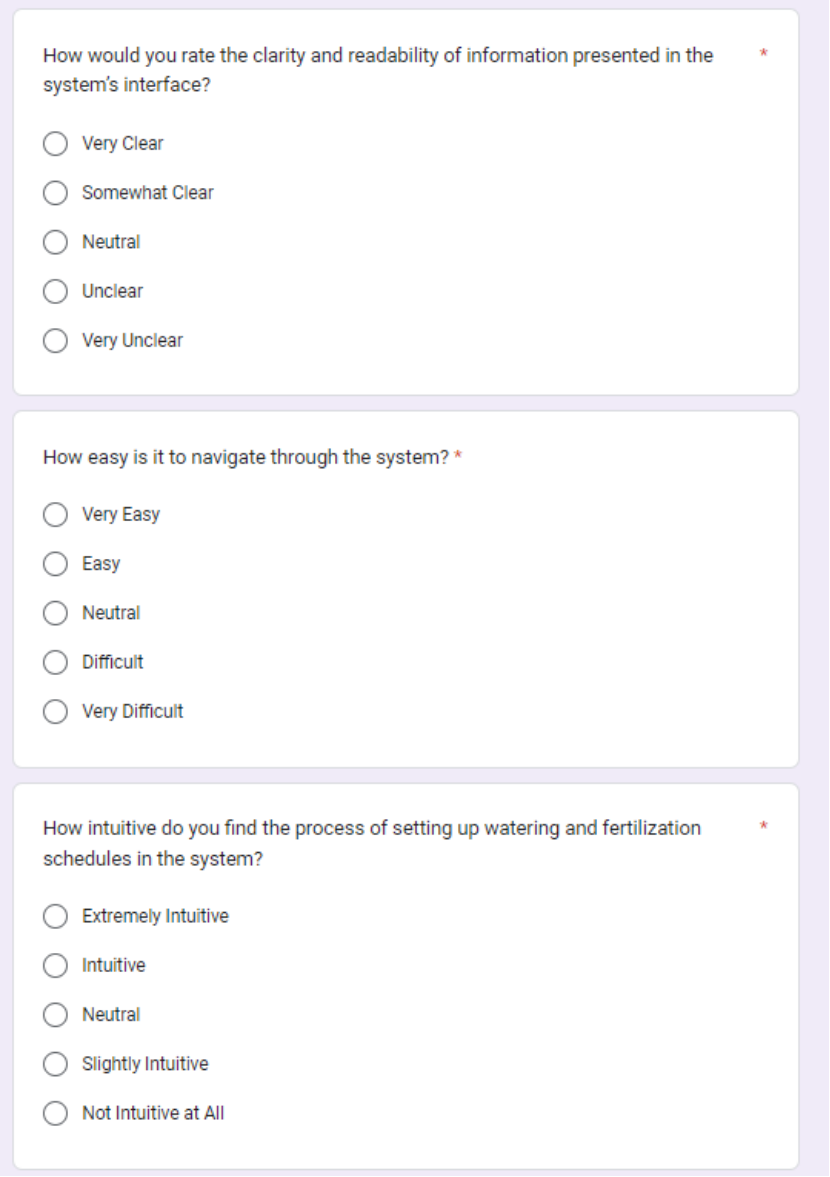


Figure 5.2: Survey - Google Form page2

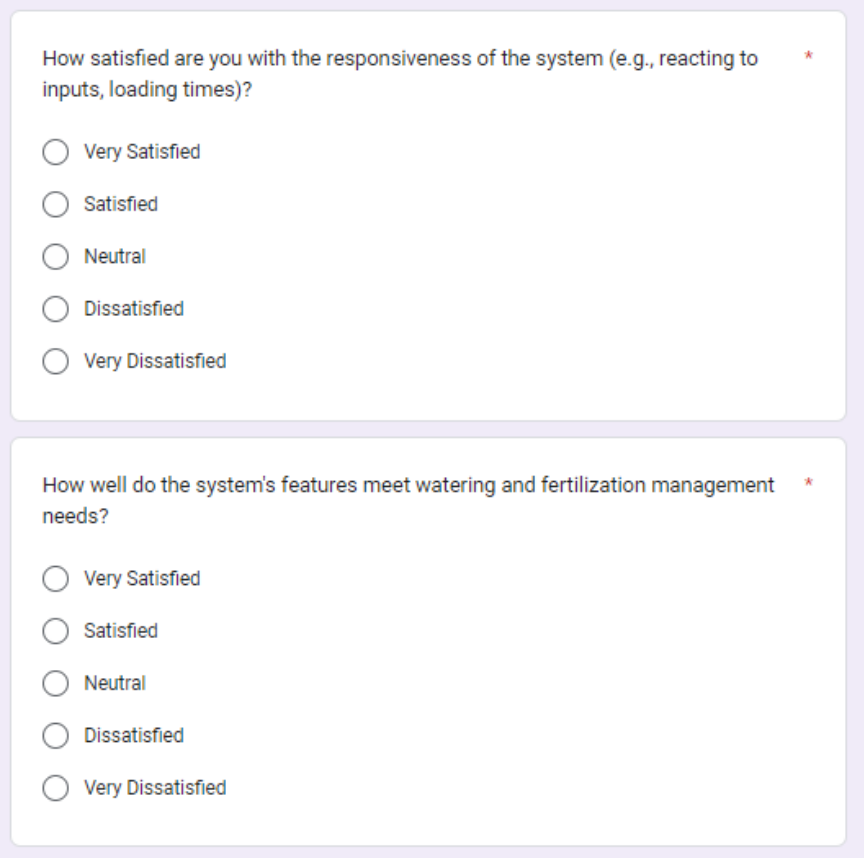


Figure 5.3: Survey - Google Form page3

5.4.2. Results of the Survey

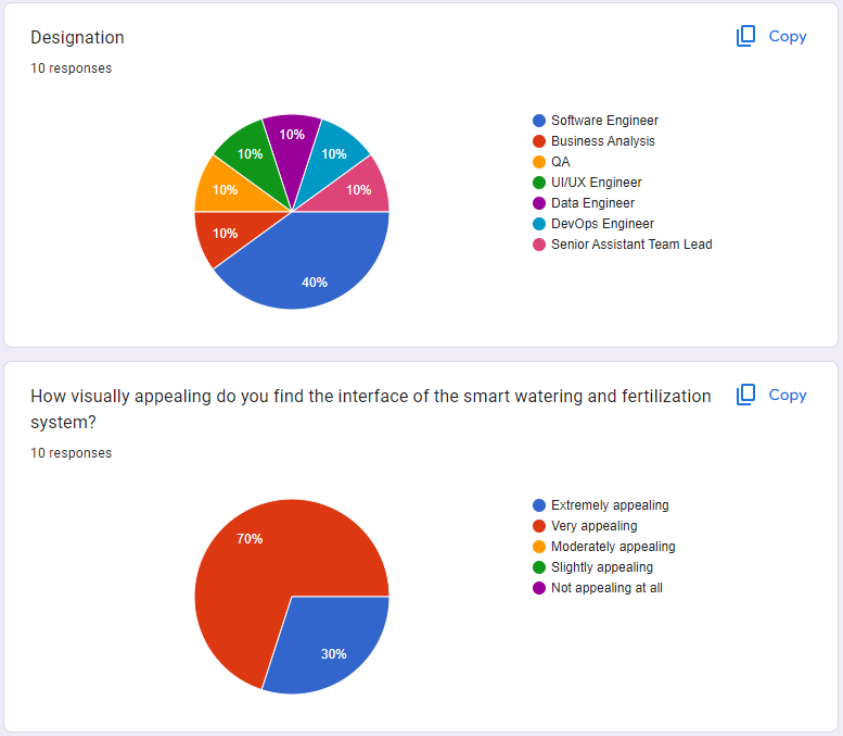


Figure 5.4 Survey Result Page 1

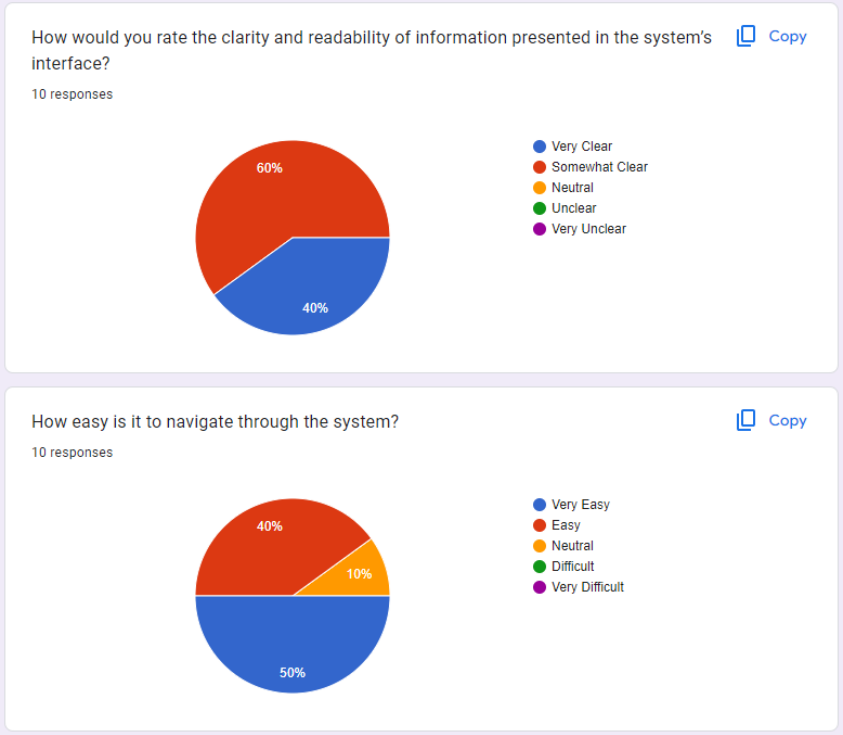


Figure 5.5 Survey Result Page 2

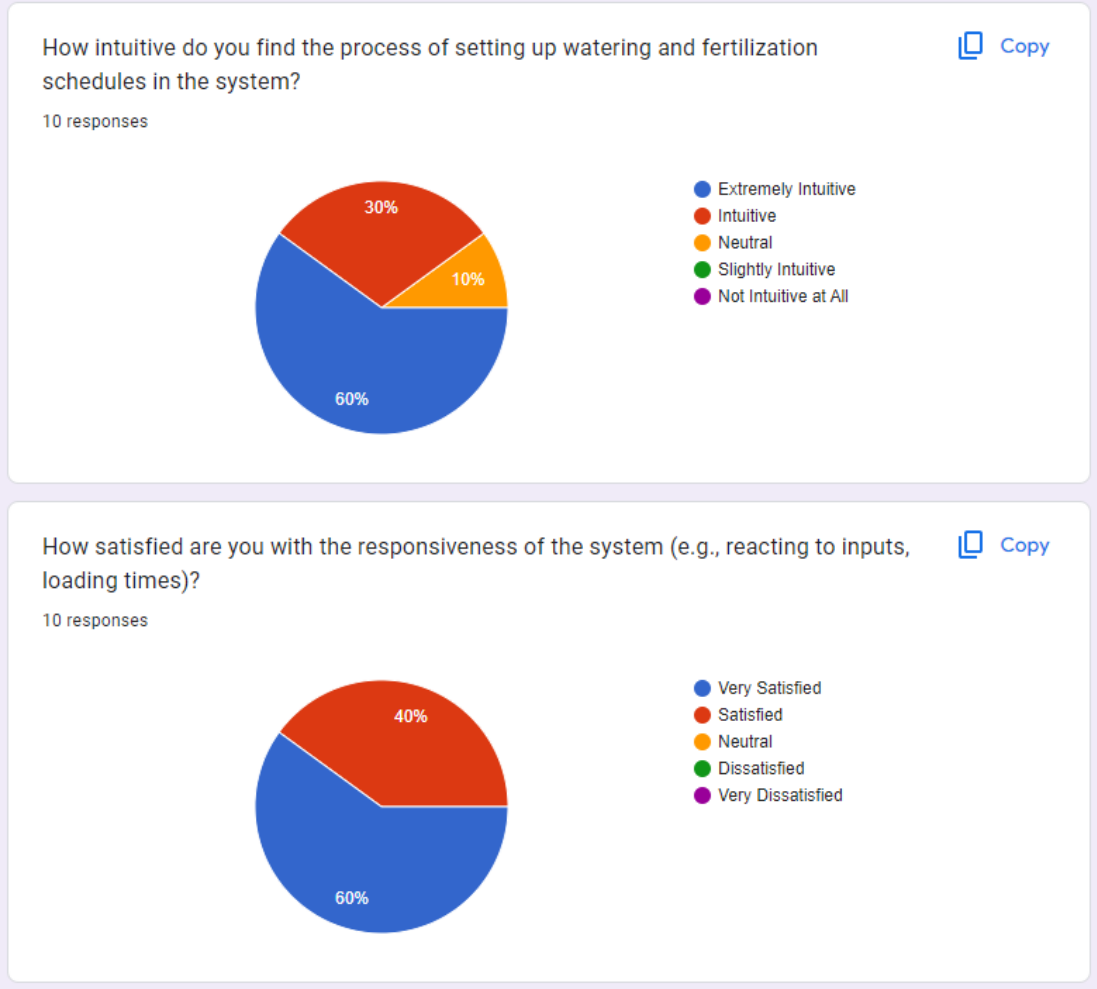


Figure 5.6 Survey Result Page 3

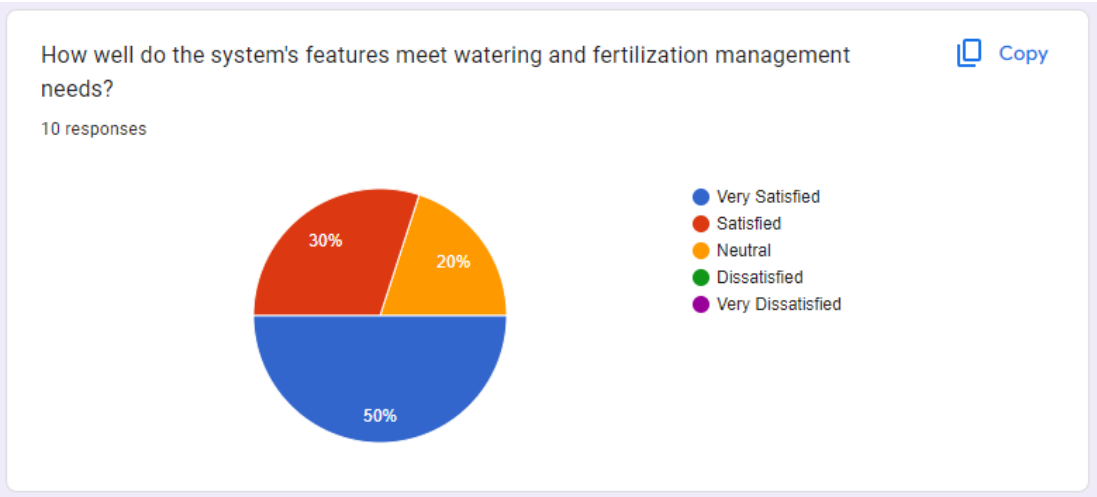


Figure 5.7 Survey Result Page 4

5.4.3. Small-scale pilot project

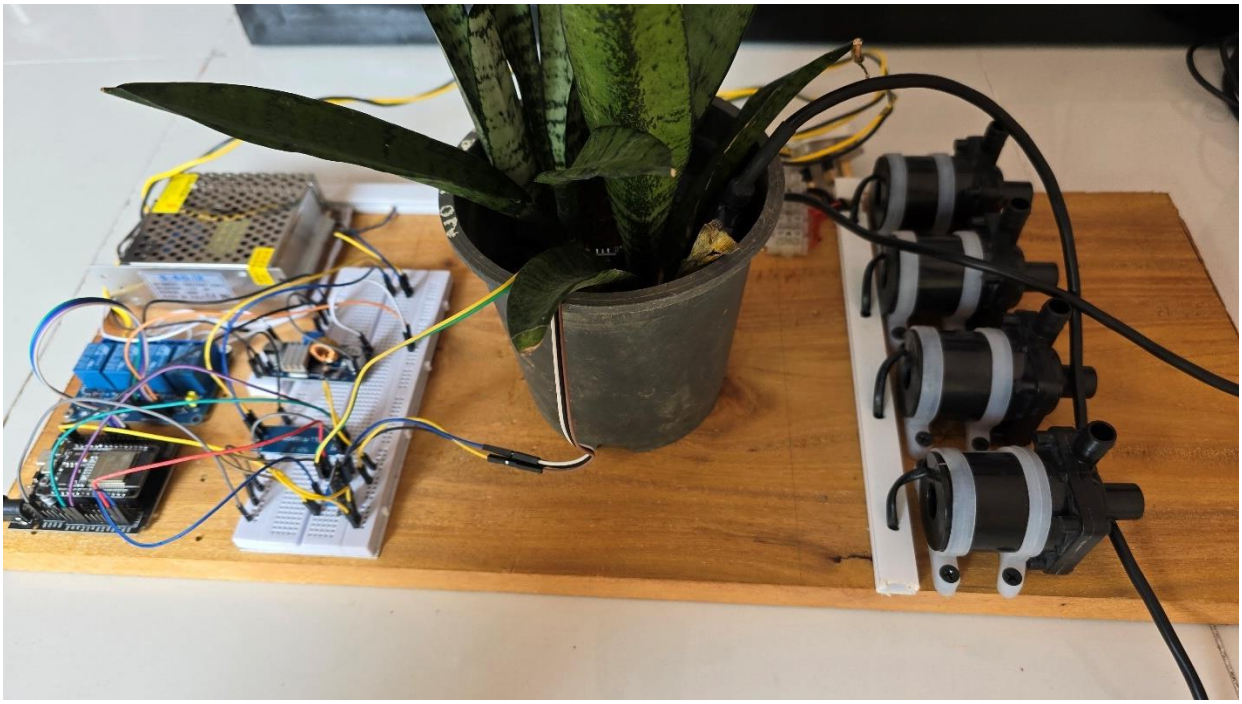


Figure 5.8 Hardware component applied to small plant

During this evaluation phase, the system was implemented in a specific agricultural environment as shown in figure 5.4 and its operation and performance could be monitored closely for a specified period. The aim was to collect actual information from real-world scenarios in order to establish an accurate basis for evaluating the functionality of the system. Watering process evaluated in with the pilot project.

5.4.4. Data Collection and Analysis

Throughout the pilot project, few metrics were recorded in the database. The metrics covered water utilization, fertilizer utilization, and sensor reading values. Following this, an evaluation was conducted on the gathered data in order to determine the functional accuracy and overall efficacy of the system.

5.4.5. Results and Adjustments

Based on the data recorded, a few changes have been applied for calculations in watering and fertilization algorithms. Conclusions are made based on the following results.

	Soil Moisture	Temperature	Humidity	Water Amount
Day 1	20	32	80%	78ml
Day 2	31	31.3	82%	36ml
Day 3	34	30	95%	39ml
Day 4	32	30	75%	18ml
Day 5	30	30	75%	31ml

Table 5.5 Pilot Project Results

Based on the results, The smart water and fertilizing system optimized water and fertilizer usage by reducing waste and ensuring resources were applied in the most effective amounts. This was achieved by closely monitoring and making automated modifications based on real-time soil and environmental variables. Optimal application of water and fertilizers can positively impact crop health and productivity.

Based on the results, the soil moisture value has stable within a small range of values from day 2 onwards. Based on that, we can conclude that the soil's water level has remained constant.

6 CHAPTER – CONCLUSION

Smart IoT based water and fertilization system allowed to find feasibility of implementing IoT technologies and components into plants and optimize water and fertilizer resources. This system is innovative development in precision agriculture in Sri Lanka. System offers prediction based on data analysis. System capable of monitoring plants and process watering and fertilization in both manual and automated ways. Monitoring includes soil moisture, temperature, humidity and NPK level of the plant. System developed with the latest technologies with the industry standards in implementation. Based on the trained machine learning model system provide few predictions for sensor values. System ensure that the plants are consuming resources for its required level and it will effectively resolve primary causes of water and fertilizer wastage.

6.1. Future Improvement

Mobile App – Develop a mobile application to increase the usability.

Enhance Capacity – Increase system capability to handle wide range of plants with increasing IoT components.

Rain Water Utilization – Utilize rainwater by harvesting and recycle method to minimize water usage.

Improve Machine Learning Models – Train additional models from the system's past data and produce more model-based predictions.

6.2. Knowledge Gained

I was able to explore wide range of technologies, best practices in coding, industry standards with this project. This project allowed me to apply previously learned lessons to a real-world scenarios. It’s been a full of educational effort that allow me to gain my knowledge. Project inspire me to learn advanced concept of software engineering and apply those concepts into the system.

7 REFERENCES

Chandrasekara, S.S.K. et al, (2021). *A review on water governance in Sri Lanka.* [Online]
Available at: https://iwaponline.com/wp/article/23/2/255/80096/A-review-on-water-governance-in-Sri-Lanka-the
[Accessed: 21 October 2023].

Rodrigo, C., (2013). *Will Sri Lanka run out of water for agriculture or can it be managed.* [Online]
Available at: https://www.ips.lk/talkingeconomics/2013/03/22/will-sri-lanka-run-out-of-water-for-agriculture-or-can-it-be-managed
[Accessed: 21 October 2024].

Thibbotuwawa, M., (2021). *Why the transition to smart farming is critical in Sri Lanka.* [Online]
Available at: https://development.asia/insight/why-transition-smart-farming-critical-sri-lanka
[Accessed: 15 November2024].

Bisht, Chauhan, (2020). *Excessive and disproportionate use of chemicals cause soil contamination and nutritional stress*. [Online]
Available at: https://www.intechopen.com/chapters/74460
[Accessed: 21 November2024].

Ashwini (2018). *A study on smart irrigation system using IOT for surveillance, Research Gate*. [Online]
Available at:
https://www.researchgate.net/publication/327964370_A_Study_on_Smart_Irrigation_System_Using_IoT_for_Surveillance_of_Crop-Field
[Accessed: 21 November 2024].

Amalraj, Banumathi, John,.(2019). *A study on smart irrigation systems for agriculture using IOT*, *International Journal of Scientific & Technology Research*. [Online]
Available at: https://www.ijstr.org/final-print/dec2019/A-Study-On-Smart-Irrigation-Systems-For-Agriculture-Using-Iot-.pdf
[Accessed: 12 January 2024].

Hamoodi, N. Hamoodi, M. Haydar,. (2020). *Automated irrigation system based on soil moisture using Arduino board*, *Bulletin of Electrical Engineering and Informatics*.[Online]
Available at: https://beei.org/index.php/EEI/article/view/1736/1464
[Accessed: 15 January 2024].

Joshi, Raval, Patel,. (2021). D*esign and implementation of a smart irrigation system for Improved Water-Energy Efficiency* [Online]
Available at:
https://www.researchgate.net/publication/318448984_Design_and_Implementation_of_a_Smart_Irrigation_System_for_Improved_Water-Energy_Efficiency
[Accessed: 25 January 2024].

John Deere.(2024). *John Deere US | Products & Services Information*. [Online]
Available at: https://www.deere.com/en/
[Accessed: 21 February 2024].

GroGuru. (2024) *Strategic Water Management Solutions for Farmers.* [Online]
Available at: https://www.groguru.com/
[Accessed: 21 February 2024].

Growlink. (2024). *Smart irrigation system.* [Online]
Available at: https://www.growlink.ag/
[Accessed: 21 February 2024].